

北京理工大学

本科生毕业设计（论文）

代理内核操作系统实验 PKE 在 K210 开发板
上的移植和改进

Transplantation and improvement of Proxy Kernel operating
system experiment (PKE) on K210 board

学 院：	计算机学院
专 业：	计算机科学与技术
学生姓名：	张国安
学 号：	1120181447
指导教师：	陆慧梅

2022 年 5 月 18 日

原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名：

日期：

年

月

日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名：

日期：

年

月

日

指导老师签名：

日期：

年

月

日

代理内核操作系统实验 PKE 在 K210 开发板上的移植和改进

摘 要

本文……。

摘要正文选用模板中的样式所定义的“正文”，每段落首行缩进 2 个字符；或者手动设置成每段落首行缩进 2 个汉字，字体：宋体，字号：小四，行距：固定值 22 磅，间距：段前、段后均为 0 行。阅后删除此段。

摘要是一篇具有独立性和完整性的短文，应概括而扼要地反映出本论文的主要内容。包括研究目的、研究方法、研究结果和结论等，特别要突出研究结果和结论。中文摘要力求语言精炼准确，本科生毕业设计（论文）摘要建议 300-500 字。摘要中不可出现参考文献、图、表、化学结构式、非公知公用的符号和术语。英文摘要与中文摘要的内容应一致。阅后删除此段。

关键词：北京理工大学；本科生；毕业设计（论文）

Transplantation and improvement of Proxy Kernel operating system experiment (PKE) on K210 board

Abstract

In order to study……

Abstract 正文设置成每段落首行缩进 2 字符，字体：Times New Roman，字号：小四，行距：固定值 22 磅，间距：段前、段后均为 0 行。阅后删除此段。

Key Words: BIT; Undergraduate; Graduation Project (Thesis)

目 录

摘 要	I
Abstract	II
第 1 章 绪论	1
1.1 研究工作的背景和意义	1
1.2 国内外研究现状和发展态势	1
1.3 本文的主要贡献与创新	1
1.4 本文的结构安排	1
第 2 章 理论基础及相关技术	2
2.1 代理内核操作系统实验	2
2.1.1 代理内核的概念	2
2.1.2 代理内核的思想	2
2.1.3 代理内核实验的基本介绍	3
2.2 RISC-V 新型开放指令集和精简指令集介绍	3
2.2.1 RISC-V 的基本介绍	3
2.2.2 RISC-V 的特权架构	3
2.2.3 RISC-V 的中断、异常委托	4
2.3 现有内核在 K210 移植工作的参考	4
2.3.1 K210 的基本信息	4
2.3.2 THU ucore 在 K210 的移植	5
2.3.3 MIT xv6 在 K210 的移植	5
第 3 章 移植 PKE 的总体设计	6
3.1 移植 PKE 到 K210 的背景	6
3.1.1 PKE 移植前开发环境介绍	6
3.1.2 PKE 移植前开发环境的优缺点	7
3.1.3 PKE 移植后的预期收益	8
3.2 移植 PKE 的目标	8
3.3 移植 PKE 的需求分析	9
3.4 移植 K210 前 PKE 的总体设计	9
3.4.1 系统架构	9
3.4.2 主要功能模块介绍	9
3.4.3 执行流程	9

3.5 移植 PKE 到 K210 的技术方案	9
3.6 移植 K210 后 PKE 的总体设计	9
3.6.1 系统架构	9
3.6.2 主要功能模块介绍	9
3.6.3 执行流程	9
第 4 章 环境搭建	10
4.1 软件环境	10
4.1.1 编译工具链	10
4.1.2 代码准备	11
4.1.3 编辑器、IDE 选择	12
4.1.4 K210 环境	12
4.1.5 自动化编译脚本	13
4.2 硬件环境	13
4.2.1 K210 硬件要求	13
第 5 章 移植 PKE 的具体实现	14
5.1 引入 RustSBI	14
5.1.1 SBI 背景与现状	14
5.1.2 RustSBI 的中断和异常委托	14
5.1.3 RustSBI 提供 Bootloader 功能与运行时服务	14
5.1.4 使用 RustSBI 兼容 K210 旧版指令集	14
5.1.5 使用 RustSBI 兼容不同的 RISC-V 开发板	15
5.2 编译流程改造	15
5.2.1 编译流程改造的背景	15
5.2.2 内存布局改造	15
5.2.3 Makefile 改造	16
5.2.4 编译自动化脚本编写	17
5.3 内核启动流程改造	17
5.3.1 内核启动流程改造前后对比	17
5.3.2 用户程序加载	17
5.3.3 内核程序入口点修改	18
5.3.4 内核初始化	18
5.4 驱动开发	18
5.4.1 串口驱动	18
5.4.2 时钟驱动	18

5.5	HTIF 依赖移除及接口移植	18
5.5.1	涉及接口梳理	18
5.5.2	接口移植的技术方案及实现	19
5.6	用户程序加载	22
5.6.1	单个进程加载	23
5.6.2	多进程支持	23
第 6 章	代理内核实验的参考实现	24
6.1	系统调用	24
6.1.1	实验预期	24
6.1.2	具体实现	25
6.1.3	执行流程	25
6.2	异常处理	25
6.2.1	实验预期	25
6.2.2	实验分析	25
6.3	定时器中断	25
6.3.1	实验预期	25
6.3.2	实验改进	25
6.3.3	具体实现	25
6.3.4	执行流程	25
6.4	虚拟地址和物理地址的转换	25
6.4.1	实验预期	25
6.4.2	具体实现	25
6.4.3	执行流程	25
6.5	基本的内存管理	25
6.5.1	实验预期	25
6.5.2	具体实现	25
6.5.3	执行流程	25
6.6	栈空间不足与缺页异常	25
6.6.1	实验预期	26
6.6.2	具体实现	26
6.6.3	执行流程	26
6.7	创建子进程 fork 的实现	26
6.7.1	实验预期	26
6.7.2	兼容 K210 与改进	26

6.7.3 具体实现	26
6.7.4 执行流程	26
6.8 进程的控制权交接	26
6.8.1 实验预期	26
6.8.2 具体实现	26
6.8.3 执行流程	26
6.9 进程的时间片调度	26
6.9.1 实验预期	26
6.9.2 具体实现	26
6.9.3 执行流程	26
6.10 实验指导书的编写及管理	26
6.10.1 实验指导书的编写	26
6.10.2 实验指导书的管理	26
6.10.3 对应代码库的管理	26
结 论	27
参考文献	28
附 录	30
附录 A L ^A T _E X 环境的安装	30
附录 B BITHesis 使用说明	30
致 谢	31

第 1 章 绪论

- 1.1 研究工作的背景和意义**
- 1.2 国内外研究现状和发展态势**
- 1.3 本文的主要贡献与创新**
- 1.4 本文的结构安排**

第 2 章 理论基础及相关技术

2.1 代理内核操作系统实验

2.1.1 代理内核的概念

代理内核 (Proxy Kernel) 是一种特殊的操作系统内核。代理内核系统是由代理内核和 Host 主机的操作系统 Ubuntu 组成的。代理内核与 Host 主机的操作系统之间使用 HTIF 接口进行通信。

代理内核并不是一个独立的操作系统，它虽然拥有 I/O 功能，但它不具备 I/O 的独立实现。它的 I/O 功能实现依赖于 Host 主机的操作系统 Ubuntu。也就是说，代理内核 Proxy Kernel 与 Host 主机的操作系统 Ubuntu 是并行在运行的。它们之间通过 HTIF(Host Target Interface) 通信。当代理内核需要进行 I/O 时，代理内核就通过 HTIF 调用 Host 主机的操作系统 Ubuntu 的 I/O 接口，以达到 I/O 的目的。

2.1.2 代理内核的思想

代理内核是操作系统的核心。它只关心内存和 CPU 的管理，不关心外部设备 I/O 功能的实现。代理内核的 I/O 功能都代理给 Host 主机的操作系统 Ubuntu。这样做的好处是，开发者不必在拘泥于繁琐的外部设备的 I/O 实现，而是将注意力集中于计算机的核心资产 CPU、内存等。

除此之外，因为代理内核少了具体的 I/O 实现，代理内核的代码会变得加更精简。精简的代理内核代码更便于维护，也更便于后续学习者学习。

代理内核还有助于我们快速验证 CPU 软核。由于代理内核只关心 CPU 的管理，内存的管理。这么一来，我们就可以将代理内核运行在更加独立的 RISC-V CPU 软核上，该 CPU 软核不需要实现 I/O 功能。我们也可以更加集中精力去验证 CPU 软核的功能。从而进行快速的迭代和开发。在真实的物理环境 Zedboard 开发板上，代理内核的代码会被编译成 RISC-V 指令，最终运行在 FPGA 上的 RISC-V CPU 软核上。与此同时，Zedboard 开发板上的 ARM 物理核运行着 ARM 版本的 Ubuntu 操作系统。代理内核通过 Zedboard 上的 HTIF 接口调用 Ubuntu 的 I/O 接口，以达到 I/O 的目的。这种设计下，我们不需要实现 CPU 的 I/O 功能，我们可以快速对 RISC-V CPU 软核进行验证。

2.1.3 代理内核实验的基本介绍

代理内核实验 PKE 主要分为三个部分: a. 系统调用、异常处理、时钟中断, b. 内存管理, c. 进程管理。后面的实验对前面的实验具有依赖关系, 如果前面的实验没有完成, 后面的实验就不能进行。每个实验都具有用户态程序、内核态程序。实验操作者需要根据用户态程序的需求, 补全内核代码。只有这样, 代理内核自身和用户程序才能正常运行。

代理内核实验 PKE 的每个小实验都较为精简紧凑。对于实验操作者来说, 在梳理好每个实验执行流程之后, 就可以很容易对实验代码进行补全。实验代码短小精悍, 行数虽然少, 但是代码的意义却很大。它可以帮助实验操作者集中精力于管理计算机的内存、CPU, 帮助实验操作者快速验证操作系统的理论知识。

2.2 RISC-V 新型开放指令集和精简指令集介绍

2.2.1 RISC-V 的基本介绍

RISC-V 指令集是一种新型的开放和精简指令集。它通过取长补短的方式, 吸取了以往 CPU 指令集的优点、避免了以往 CPU 指令集的大部分缺点。

它是一个基于开放架构的指令集, RISC-V 的指令是开放的, 任何人都可以根据这个开放的指令集去实现 RISC-V 架构的 CPU。因此, RISC-V 指令集具有开源、免费的特点。

它是一个精简的指令集。RISC-V 手册指出, 同样的 C 语言程序, 经过适用于不同 CPU 架构的编译器编译后, 得到的指令序列中, RISC-V 架构的指令是相对来说数量较少、较精简的。这对减少嵌入式设备功耗具有重要的意义。

2.2.2 RISC-V 的特权架构

RISC-V 的特权架构主要分为三种模式: 机器模式 (Machine-mode)、监管者模式 (Supervisor-mode)、用户模式 (User-mode)。

M 模式是拥有最高级权限的模式, M 模式的硬件线程完全掌控着寄存器、内存、I/O 的控制权。M 模式具备拦截和处理异常与中断的能力。除此之外, 它还可以将异常与中断托管给 S 模式。

S 模式的产生, 主要是为了解决存储碎片化的问题。S 模式之下, 我们可以开启分页机制, 使用基于页面的虚拟内存。这种模式的特权级相比 M 模式较低。与 M 模

式相比，S 模式拥有的权限较少。与 U 模式相比，S 模式拥有的权限较多。我们的操作系统内核 PKE，主要就是运行在 S 模式。

U 模式也就是用户态程序运行时的模式。它具备最低的特权级，拥有的权限最少。它所需的许多功能，都需要通过系统调用、中断、异常等机制获取。它通过这些机制，让 CPU 的执行切换到更高特权级的内核中，让内核来提供并执行用户态程序所需功能（如串口输出、定时器、异常处理等）。

这三种模式之间不是普通的链式关系，而是一层叠一层的金字塔结构。更高特权级的底层为上层提供运行时服务，更低特权级的上层运行并依赖于底层之上。

2.2.3 RISC-V 的中断、异常委托

前面我们提到，M 模式具备拦截和处理异常与中断的能力。M 模式下运行的内核可以根据中断/异常的状态码和向量表选择对应的处理程序。拦截和处理中断、异常。

除此之外，M 模式还可以将这两者的拦截和处理委托给 S 模式的内核。M 模式的内核可以在初始化时，通过设置异常委托寄存器（`medeleg`）、中断委托寄存器（`mideleg`），将异常和中断的拦截与处理委托给 S 层。M 层的内核只需要在初始化时，根据异常、中断的状态码值，在委托寄存器的对应二进制位赋值即可。委托寄存器的初始化结束以后，寄存器上对应的异常/中断的拦截与处理都会交接给 S 模式的内核。

2.3 现有内核在 K210 移植工作的参考

2.3.1 K210 的基本信息

K210 是由嘉楠公司推出的一款 MCU，它具有成本低、高性能、性价比高等特点。

K210 拥有基于 RISC-V 架构、64 位的双核 CPU。它具有 6MB 物理内存，它的晶振频率是 400-600MHZ。除此之外，它拥有嘉楠自研的神经网络计算加速器 KPU，可以进行高性能的神经网络计算。

K210 的成本低廉。一块不带其他外设的 K210 MCU，只需要一百多元。这样成本低廉的物理开发板对嵌入式开发、操作系统实验教学、学生自学计算机知识很有好处。



图 2-1 K210 MCU

2.3.2 THU ucore 在 K210 的移植

ucore 是清华大学基于 MIT 开源的 xv6 内核开发的操作系统内核，它可以运行在 X86 计算机上，也可以运行在 X86 硬件模拟器上，如 QEMU、VirtualBox 等。麻雀虽小，五脏俱全。ucore 的代码量不到 5000 行，却包含了虚拟内存管理、进程管理、文件系统等主要内核功能。

早在 2020 年，南开大学的操作系统团队发起了 ucore 在 RISC-V 指令集 CPU 的移植活动。从 2020 年 3 月到 2020 年 4 月，历时 4 个月，他们完成了 ucore 从 X86 物理机到 RISC-V 物理机的移植，最终让 ucore 成功运行在 RISC-V 架构的 K210 上。

我们可以参考南开操作系统团队的移植工作，来完成 PKE 的移植与改进。

2.3.3 MIT xv6 在 K210 的移植

MIT 的 Frans Kaashoek 等在 2006 年参考 PDP-11 上的 UNIX Version 6 写了一个可在 X86 上跑的操作系统 xv6（基于 MIT License）

华科的操作系统团队完成了 xv6 从 X86 物理机到 RISC-V 物理机的移植，最终让 xv6 成功运行在 K210 上。

同理，我们可以参考华科操作系统团队的移植工作，来完成 PKE 的移植与改进。

第3章 移植 PKE 的总体设计

3.1 移植 PKE 到 K210 的背景

接下来，本文将通过阐述 PKE 移植前的开发环境与移植后的预期收益，介绍移植 PKE 到 K210 的背景。

3.1.1 PKE 移植前开发环境介绍

PKE 移植前的运行环境主要分为两种，分别是硬件环境和模拟器环境。他们的总体情况是一致的，但还是有一些细微的区别。

在模拟器环境上，PKE 运行在 spike 模拟的 RISC-V 64 位机器上。而 spike 模拟器则运行在宿主机 Ubuntu 上。当 PKE 需要使用外部设备的 I/O 功能时，PKE 就通过调用 spike 提供的 HTIF 接口，让 Ubuntu 提供 I/O 功能，以访问宿主机上的外设资源。

如下图，本文给出了模拟器环境的系统架构。

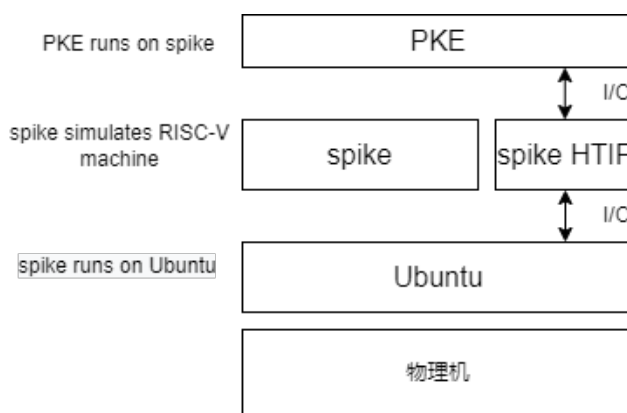


图 3-1 模拟器环境架构

在硬件环境上，我们的系统是运行在 PYNQ-Z2 开发板上。PYNQ-Z2 开发板是 Xilinx 公司的一款开发板，它具有丰富的外设资源、强大的性能。此外，它还兼容了 Arduino 和树莓派接口。

PYNQ-Z2 搭载了 FPGA 系统和 ARM 处理器。FPGA 可以烧录 RISC-V CPU 软核，ARM 硬核可以运行 Ubuntu 操作系统。我们的代理内核 PKE 运行在 RISC-V CPU 软核上，而提供外设功能的 Ubuntu 则运行在 ARM 硬核上。PKE 和 Ubuntu 之间是通过 riscv-fesvr 通信的。riscv-fesvr 是 PYNQ 开发板上的重要工具，没有它，PKE 就无

法调用 ARM 上的 Ubuntu I/O 接口。riscv-fesvr 通过共享内存的方式，接收 PKE 的请求，使用 ARM 端的资源，并向 PKE 提供资源与功能。

如下图，本文给出了硬件环境的系统架构。

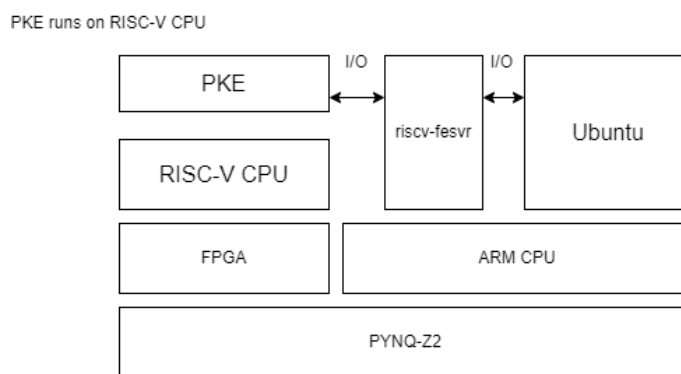


图 3-2 硬件环境架构

3.1.2 PKE 移植前开发环境的优缺点

在上一小节，本文简单介绍了 PKE 在移植前的开发环境。在本小节，本文会客观分析 PKE 原先的物理开发环境的优点与缺点。

在原先的物理环境中，代理内核的系统是运行在 PYNQ-Z2 板子上的。PYNQ-Z2 拥有 FPGA 和 ARM 物理核。FPGA 可以让开发者烧录 RISC-V CPU 软核，然后让开发者在 RISC-V CPU 软核上运行 PKE。ARM 硬核可以运行 Ubuntu 操作系统，提供 I/O 功能。

从不同的开发方向看。如果开发者是在开发 CPU 软核，需要快速验证 FPGA 上的 CPU 软核并进行迭代。这种物理环境允许开发者不实现 CPU 涉及 I/O 的功能，便于开发快速开发和验证 CPU 软核。

如果开发者是在开发或学习操作系统，这种开发环境反而对开发者不利。一个原因是提供这种物理环境的开发板 PYNQ-Z2 价格昂贵，成本高昂。其价格已经快要接近一台 PC 机的价格。此开发板在某网购平台上的均价已达到三千余元。这对入门级操作系统学习者购买资料学习、课程实验教学都是不利的。

另一个原因环境配置繁琐，开发者需要在 PYNQ 上烧录 RISC-V CPU 软核，还需要在 ARM 核上部署 Ubuntu。除此之外，还需要在 ARM 核上部署 riscv-fesvr，让 PKE 与 Ubuntu 通信。这种开发环境的部署成本和学习成本是很高的，这对于刚入门

操作系统实验学习的开发者是不利的。

总而言之，PKE 系统的物理环境价格昂贵、成本高昂、搭建成本高、学习成本高，不利于学习者入门，也不利于大规模展开实验教学。这些是 PKE 移植前的物理环境的痛点。为了解决这些痛点，本文提出了将 PKE 移植到 K210 上的方案。将在下一小节阐述该方案的收益。

3.1.3 PKE 移植后的预期收益

上一小节我们提到，PKE 原先的物理环境有许多缺点。这些缺点不利于学习者入门、也不利于大规模展开实验教学。因此，我们计划将 PKE 移植到 K210 物理环境上。PKE 移植后，我们将拥有如下收益：

1. 成本低廉

K210 板子价格低廉。一块不带外设的普通 K210 SoC 的市场价格只需要一百多元。与动辄就要三千余元的 PYNQ-Z2 相比，K210 的经济成本仅仅是 PYNQ-Z2 的三十分之一。这非常有利于我们开展大规模的物理环境实验教学。也非常有利于入门学习者购买资料进行学习。

2. 物理环境搭建成本低

在 K210 上，我们不再像从前在 PYNQ-Z2 那样烧录 RISC-V CPU 软核，也不需要再在 PYNQ-Z2 的 ARM 核上加载运行 Ubuntu。由于 K210 带有 RISC-V 64 位 CPU，我们只需要在 K210 上烧录并运行 PKE 即可。这使得我们的物理环境搭建成本变得极低。

3. 环境搭建学习成本低

移植 PKE 到 K210 后，我们可以使用 docker 配置环境包、编写 Makefile 将编译与烧录流程变得自动化。由于不再需要像在 PYNQ-Z2 上烧录 RISC-V CPU 软核，也不需要再在 PYNQ-Z2 的 ARM 核上加载运行 Ubuntu。我们的环境搭建是自动化的。环境搭建的学习成本也是很低的。

3.2 移植 PKE 的目标

1. 用户态程序无感知

2. 减少实验操作者对移植的感知

3. 降低 PKE 在物理环境开发的成本
4. 提高实验操作者的开发效率
5. 降低 PKE 后续移植的成本

3.3 移植 PKE 的需求分析

3.4 移植 K210 前 PKE 的总体设计

3.4.1 系统架构

3.4.2 主要功能模块介绍

3.4.3 执行流程

3.5 移植 PKE 到 K210 的技术方案

3.6 移植 K210 后 PKE 的总体设计

3.6.1 系统架构

3.6.2 主要功能模块介绍

3.6.3 执行流程

第 4 章 环境搭建

4.1 软件环境

4.1.1 编译工具链

step1. 访问 sifive 官网，下载 riscv gcc toolchain

<https://www.sifive.com/software>

step2. 找到 Prebuilt RISC-V GCC Toolchain。根据开发环境选择对应的版本。

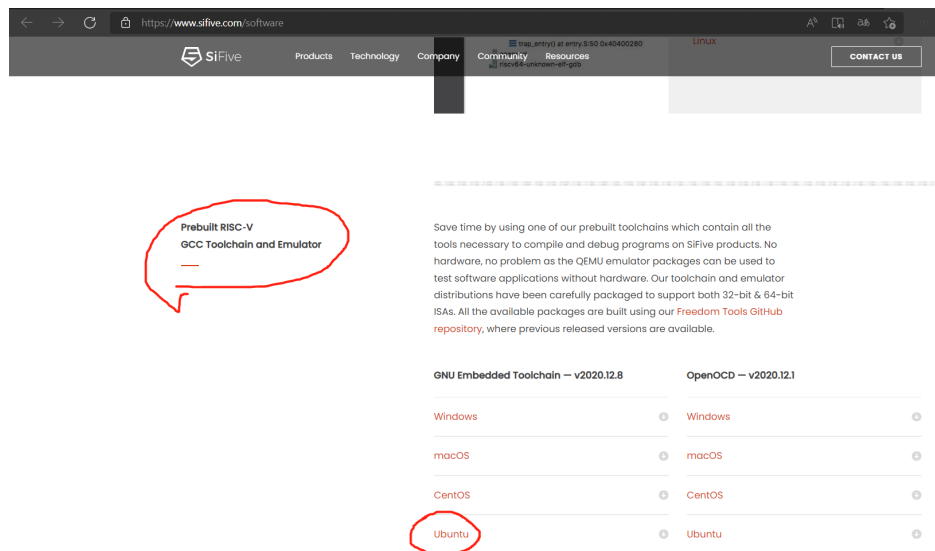


图 4-1 编译工具链列表

step3. 将下载好的 tar.gz 压缩包解压

```
1 tar -zxvf $your_tar_gz
```

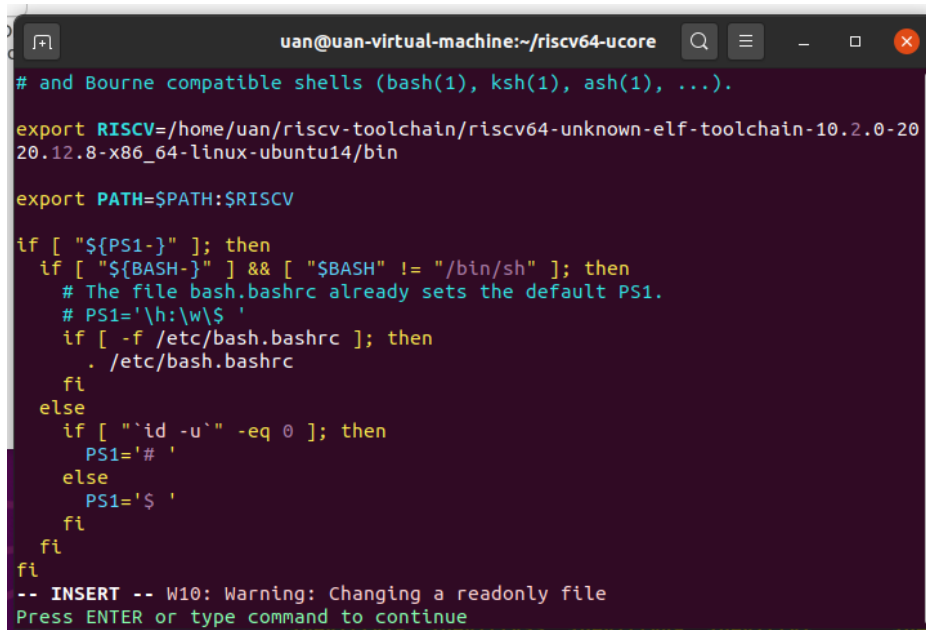
代码 4-1: 解压命令

step4. 配置环境变量。解压完成得到文件夹，进入文件夹里的 bin 目录，打开 terminal，输入 pwd 获得当前路径。复制获得的路径。将复制到的路径加入系统的 PATH 环境变量。

```
1 vim /etc/profile
2 #添加以下两行到文件末尾
3 export RISCVC=$your_path
```

```
4 export PATH=$PATH:$RISCV
```

代码 4-2: 修改环境变量



A terminal window titled 'uan@uan-virtual-machine:~/riscv64-ucore' showing the configuration of environment variables. The terminal output includes comments about Bourne compatible shells, the export of RISCV to the toolchain path, the export of PATH to include RISCV, and a complex if statement for setting the PS1 prompt. At the bottom, there is a warning message: '-- INSERT -- W10: Warning: Changing a readonly file Press ENTER or type command to continue'.

```
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

export RISCV=/home/uan/riscv-toolchain/riscv64-unknown-elf-toolchain-10.2.0-20
20.12.8-x86_64-linux-ubuntu14/bin

export PATH=$PATH:$RISCV

if [ "${PS1-}" ]; then
  if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi

-- INSERT -- W10: Warning: Changing a readonly file
Press ENTER or type command to continue
```

图 4-2 配置环境变量

step5. 加载环境变量文件

```
1 source /etc/profile
```

代码 4-3: 加载环境变量文件

step6. 验证编译环境

在终端输入 riscv64-unknown-elf-gcc -v，如果出现以下内容，则编译环境配置成功。

```
1 Using built-in specs.
2 ....
3 gcc version 10.2.0 (SiFive GCC-Metal 10.2.0-2020.12.8)
```

代码 4-4: 验证编译环境

4.1.2 代码准备

step1. 下载 riscv64-pke-k210 的代码库并查看所有分支

```
1 git clone git@github.com:BITzga/riscv64-pke-k210.git
2 git branch -a
```

代码 4-5: 下载代码库

step2. 根据开发需求选择分支。

如下所示，k210 前缀的代码分支是根据 k210 环境移植完成的代码。而其他普通分支则是 PKE 原先的代码。此时，根据开发需求使用 `git checkout` 命令选择分支即可。

```
1 remotes/origin/k210/lab1_1_syscall
2 remotes/origin/k210/lab1_2_exception
3 remotes/origin/k210/lab1_3_irq
4 remotes/origin/k210/lab2_1_pagetable
5 remotes/origin/k210/lab2_2_allocatepage
6 remotes/origin/k210/lab2_3_pagefault
7 remotes/origin/k210/lab3_1_fork
8 remotes/origin/k210/lab3_2_yield
9 remotes/origin/k210/lab3_3_rrsched
10 remotes/origin/lab1_1_syscall
11 remotes/origin/lab1_2_exception
12 remotes/origin/lab1_3_irq
13 remotes/origin/lab2_1_pagetable
14 remotes/origin/lab2_2_allocatepage
15 remotes/origin/lab2_3_pagefault
16 remotes/origin/lab3_1_fork
17 remotes/origin/lab3_2_yield
18 remotes/origin/lab3_3_rrsched
19 remotes/origin/master
```

代码 4-6: 分支列表

4.1.3 编辑器、IDE 选择

4.1.4 K210 环境

- Python3 环境

由于烧录程序和串口调试工具都是使用 Python3 编写的。Python 是解释型语言，所以我们需要安装 Python3 解释器。除此之外我们还需要安装 Python 包管理工具 pip。

```
1 sudo apt-get install python3
2 sudo apt-get install python3-pip
3
```

代码 4-7: 安装 Python3 环境

- 烧录工具我们编写好的程序，经过编译，变成 bin 文件，还需要烧录到 K210 上才能运行。而烧录需要借助烧录工具，这里我们使用了 K-Flash。
- 串口调试工具内核在运行时，输出信息是通过串口输出的，我们需要一个串口调试工具来接收 K210 上的串口信息。这里我们需要安装并使用 miniterm。

```
1 sudo apt-get install miniterm
2
```

代码 4-8: 安装 miniterm

- RustSBI-K210 支持包 SBI 是 RISC-V 的规范之一，它规定了监管者二进制 (Supervisor Binary Interface) 接口。RustSBI-K210 是 SBI 标准的一种实现，它使用 Rust 语言进行编写，具有性能安全的特点。除此之外，RustSBI-K210 还对 K210 板子提供了特殊的支持。它还可以在 K210 上作为我们内核程序的 Bootloader。我们需要使用 RustSBI-K210 支持包来支持内核移植，这里我们需要在烧录内核时引入 RustSBI-K210 支持包。

在这里，我们可以下载到 RustSBI-K210 的 release 版本

<https://github.com/rustsbi/rustsbi-k210/releases>

4.1.5 自动化编译脚本

4.2 硬件环境

4.2.1 K210 硬件要求

硬件环境较为简单，我们只需要一块具有串口功能的 K210 板子和一根数据线即可。

第 5 章 移植 PKE 的具体实现

5.1 引入 RustSBI

5.1.1 SBI 背景与现状

移植 K210 时，PKE 需要依赖 SBI（Supervisor Binary Interface）提供 Bootloader 和 RUNTIME 功能，所以烧录内核时需要带上 SBI 固件。通过调研发现，OpenSBI 与 RustSBI（用 Rust 语言实现的 SBI）均按照 SBI 标准实现。这两种也是业内使用最多的开源 SBI。qemu 就是用了 OpenSBI 为 RISC-V 提供了环境支持。此外，RustSBI 还对 K210 做了特殊支持。所以目前暂定使用 RustSBI 当作 SBI 固件，为 PKE 提供 Bootloader 功能和 RUNTIME 运行时服务。移植过程中我们不需要关心 RustSBI 的具体实现，只需要根据 SBI 标准调用其接口即可。

除此之外，引入 SBI，可以便于内核在其他板子上运行。当更换板子时，不需要改变内核代码，只需要更改 RustSBI 的支持包，获取对应硬件平台的支持包即可。这也简化了后续内核在其他芯片的移植工作。

5.1.2 RustSBI 的中断和异常委托

5.1.3 RustSBI 提供 Bootloader 功能与运行时服务

5.1.4 使用 RustSBI 兼容 K210 旧版指令集

RustSBI 在 K210 兼容了高版本的指令。K210 实现的 RISC-V 指令集是 1.9.1 标准的。目前最新的特权级标准已经达到 1.11。如果我们的内核代码里有用到更高级的 RISC-V 汇编指令，可能会在 K210 上无法运行。这种情况下，就要改动内核的代码，会带来许多工作量。因此，使用 RustSBI 可以使我们免去处理 RISC-V 汇编版本的麻烦。

5.1.5 使用 RustSBI 兼容不同的 RISC-V 开发板

5.2 编译流程改造

5.2.1 编译流程改造的背景

5.2.2 内存布局改造

由于我们引入了 RustSBI，RustSBI 需要占用 0x80000000-0x8001FFFF 的物理内存空间。所以，内核的程序入口点由此发生了变化。我们需要修改内核 lds 文件，更改了程序的入口点以保证内核可以正常运行。

首先，我们现在 mentry.S 中加入以下两行代码，用以确保 _mentry 是内核的程序入口点。

```
1  .globl _mentry
2  .section .text.prologue, "ax"
3  _mentry:
```

代码 5-1: 修改内核程序入口点

确定了内核的程序入口点，还需要把程序入口点的地址设置为 0x80020000，这需要我们对内存布局进行改造。修改 BASE_ADDRESS，赋值为 0x80020000. 并设置代码段的起始地址为 BASE_ADDRESS，自此，内存布局就修改完成了。

```
1  OUTPUT_ARCH(riscv)
2  ENTRY(_mentry)
3  BASE_ADDRESS = 0x80020000;
4  SECTIONS
5  {
6      /*-----*/
7      /* Code and read-only segment */
8      /*-----*/
9
10     /* Begining of code and text segment, starts from DRAM_BASE to be
effective before enabling paging */
11     . = BASE_ADDRESS;
12
13     /* text: Program code section */
14     .text :
15     {
16         stext = .;
17         *(.text.prologue);
```

```

18     *(.text .stub .text.* .gnu.linkonce.t.*);
19     . = ALIGN(0x1000);
20     .....
21 }

```

代码 5-2: 修改内存布局

5.2.3 Makefile 改造

由于我们需要引入 RustSBI，并需要将其打包进入内核。除此之外，还需要将内核烧录到 K210，并与 K210 进行串口通讯。现有的 Makefile 并不支持这些工作。因此，我们需要修改 MakeFile。

```

1 $(KERNEL_K210_TARGET): $(KERNEL_TEMP_TARGET) $(BOOTLOADER)
2 $(COPY) $(BOOTLOADER) $@
3 $(V)dd if=$(KERNEL_TEMP_TARGET) of=$@ bs=128K seek=1

```

代码 5-3: 修改 Makefile

整体的编译流程为：

1. 打包内核

```

1 $(KERNEL_K210_TARGET): $(KERNEL_TEMP_TARGET) $(BOOTLOADER)
2 $(COPY) $(BOOTLOADER) $@
3 $(V)dd if=$(KERNEL_TEMP_TARGET) of=$@ bs=128K seek=1

```

代码 5-4: 打包内核

以上步骤是为了把 rust-sbi.bin 和 pke.img 打包成 kernel.img。bs=128k 意味着输入/输出的 block 大小为 128k，seek=1 意味着跳过第零个 block 进行复制操作。也就是说，内核镜像里第零个 block 存放着 rust-sbi.bin，第一个 block 才开始存放 pke.img。128k 对应着十六进制 0x20000，也就是二进制的 0010 0000 0000 0000 0000。

我们的 kernel.img 放置在 0x80000000 处，再加上以上原因，pke 的地址自然就是 0x8020000。所以，我们需要在链接脚本 kernel.lds 里指定内核起始地址为 0x8020000，这也相当于告诉 SBI 这是内核的起始地址。当 SBI 在行使 Bootloader 的功能时，会跳转到 0x8020000，将控制权转接给内核。

2. 烧录

用数据线将 K210 与上位机连接，再使用 kflash，指定好相关参数即可完成烧录。


```
1 $(PYTHON) compile_tool/kflash.py -p $(PORT) -b 1500000 $(KERNEL_K210_TARGET)
```

代码 5-5: 烧录

3. 运行 minitem, 与 K210 进行串口通讯

```
1 $(TERM) --eol LF --dtr 0 --rts 0 --filter direct $(PORT) 115200
```

代码 5-6: 运行 minitem

打开 miniterm, 接收 K210 的串口打印输出。

4. 总结

最小可执行内核在 K210 的运行流程是: 指定内核起始地址-> 打包完整内核镜像-> 烧录到 flash-> 引导程序加载和运行 RustSBI-> RustSBI 运行并跳转到内核的指定地址-> RustSBI 将控制权交接给内核-> 内核运行

5.2.4 编译自动化脚本编写

5.3 内核启动流程改造

5.3.1 内核启动流程改造前后对比

5.3.2 用户程序加载

在原先 pke 的中, 是通过调用 spike 接口, 进而调用 linux 的文件系统接口来加载用户程序的。而在 K210 上, 我们没有 spike 的环境支持, 不能直接调用 spike 接口。因此, 加载用户程序就需要自行实现文件系统, 或者使用其他办法。

由于文件系统的实现较为繁琐, 其工作量会阻塞这个移植进度。因此, 我们暂时不实现文件系统, 而是采用获取用户程序地址, 再加载的办法来实现这个需求。

具体的做法是:

1. 将用户程序、内核和 RustSBI 一起编译打包到 kernel.img
2. 使用 objdump 命令查找到用户程序 main 函数的地址
3. 得到地址以后, 把地址的值赋值到内核加载用户程序处

通过这种技术方案, 我们可以用较低的开发成本实现用户程序加载。

5.3.3 内核程序入口点修改

由于 RustSBI 已经运行在 M 态，并且为我们提供了许多运行时服务。有了 RustSBI，在 K210 上，pke 运行在 M 态会破坏 RustSBI 的设计，因此 pke 只需要运行在 S 态即可。

这样，我们就可以直接将 pke 的 M 态代码根据自身需求迁移到 S 态代码。迁移完成后，需要更改内核程序入口点至 S 态入口

修改 mentry.S 文件，将 `call m_start` 替换成 `call s_start`

5.3.4 内核初始化

5.4 驱动开发

5.4.1 串口驱动

5.4.2 时钟驱动

5.5 HTIF 依赖移除及接口移植

PKE 已有代码对 spike 提供的 HTIF（Host-Target InterFace）接口存在依赖。当 PKE 需要打印字符串到屏幕、访问主机上的文件或设备时会调用 HTIF 相关接口。PKE 通过 HTIF 接口调用 Linux 对应接口，进而实现访问外部设备的功能。

如果需要在 K210 上维持和原先 PKE 在 spike 模拟器一样的环境。K210 板子要同时运行 PKE 和 Linux 两个内核程序，还要提供 spike 模拟器类似的与 Linux 访问的 HTIF 接口。这样的移植方案的开发成本会很高。并且收益不大。

所以移植 PKE 到 K210 时，我们需要移除 PKE 对 HTIF 的依赖，自行实现其依赖功能，编写相关代码。这样 K210 只需要运行一个内核程序（PKE）即可，移植工作的开发成本会很低。

5.5.1 涉及接口梳理

串口相关接口

运行在 K210 上的 PKE，需要使用串口接口来进行串口的访问。打印字符串到上位机。这样子才能方便我们进行实验验证，程序调试。PKE 原先并未实现串口功能，而是通过调用 HTIF 接口来使用串口功能。因此，我们需要自行实现串口功能。

文件系统相关接口

原先的 PKE 没有实现文件系统，而是通过调用 HTIF 接口来使用文件系统功能。PKE 在加载用户 elf 程序文件时，通过调用 HTIF 接口来访问用户程序文件，从而将用户程序加载到内存中。最后再运行用户程序。

Device Tree 相关接口

此接口的功能是用于读取设备树文件，以屏蔽 SoC 细节。

shutdown 接口

供 panic 使用的接口，用于关闭设备。在内核 panic 时调用。PKE 原先并未实现该功能，而是通过调用 HTIF 接口来实现。

poweroff 接口

供 panic 使用的接口，用于关闭设备电源。在内核 assert 失败时使用。PKE 原先并未实现该功能，而是通过调用 HTIF 接口来实现。

定时器接口

此接口用于 PKE 设置定时中断，读取时间。PKE 中是使用 MMIO（memory mapping IO）来设置定时中断。由于 K210 开发板此方面的资料较少，不确定 K210 是否支持此方式。从减少开发成本的角度考虑，该接口使用 SBI 接口来实现的方案最低，可以免去查阅资料和测试的成本。因此，该接口也需要我们进行移植。

5.5.2 接口移植的技术方案及实现

串口及格式化输出实现

由于我们已经引入了 RustSBI，我们很容易实现串口输出。我们只需要调用 SBI 提供的串口服务即可。

```
1  uint64 SBI_CONSOLE_PUTCHAR = 1;
2  uint64 sbi_call(uint64 sbi_type, uint64 arg0, uint64 arg1, uint64 arg2) {
3  uint64 ret_val;
4  __asm__ volatile (
```

```
5  "mv x17, %[sbi_type]\n"
6  "mv x10, %[arg0]\n"
7  "mv x11, %[arg1]\n"
8  "mv x12, %[arg2]\n"
9  "ecall\n"
10 "mv %[ret_val], x10"
11 : [ret_val] "=r"(ret_val)
12 : [sbi_type] "r"(sbi_type), [arg0] "r"(arg0), [arg1] "r"(arg1), [arg2] "r"(
    arg2)
13 : "memory"
14 );
15     return ret_val;
16 }
17
18 void sbi_console_putchar(unsigned char ch) {
19     sbi_call(SBI_CONSOLE_PUTCHAR, ch, 0, 0);
20 }
```

代码 5-7: 串口实现代码

有了串口输出的方法，接下来我们需要格式化输出 `sprint.sprint` 定义在 `spike_utils.c`, 接下来我们看看 `sprint` 的代码实现。

```
1  void sprint(char *buf, const char *fmt, ...) {
2      va_list args;
3      va_start(args, fmt);
4      vsnprintf(buf, 1024, fmt, args);
5      va_end(args);
6  }
```

代码 5-8: `sprint` 实现代码

`sprint` 函数的第一个参数对应了一个字符串的起始地址，第二个参数... 代表可变参数。接下来我们点开 `vprintk` 的实现：

```
1  void vprintk(const char* s, va_list vl) {
2      char out[256];
3      int res = vsnprintf(out, sizeof(out), s, vl);
4      //you need spike_file_init before this call
5      spike_file_write(stderr, out, res < sizeof(out) ? res : sizeof(out));
6  }
```

代码 5-9: `vprintk` 实现代码

通过阅读代码发现，`vsnprintf` 并没有将字符串真正输出到控制台。而是根据原先的字符串和参数做字符串格式化，将最终结果保存在 `out` 数组中。

真正将字符串打印的函数调用是 `spike_file_write`。这个函数是调用了 `spike` 的接口，通过 `spike` 去调用 Linux 的字符串打印 API。所以我们需要在 K210 上实现串口输出，方案已经很明显，就是将 `vprintk` 函数中的 `spike_file_write` 函数替换成 `sbi_console_putchar` 实现的打印函数 `cputs`。

```
1  void vprintk(const char *s, va_list vl) {
2      char out[256];
3      int res = vsnprintf(out, sizeof(out), s, vl);
4      cputs(out);
5  }
6  /* *
7   * cputs- writes the string pointed by @str to stdout and
8   * appends a newline character.
9   * */
10 int cputs(const char *str) {
11     int cnt = 0;
12     char c;
13     while ((c = *str++) != '\0') {
14         cputch(c, &cnt);
15     }
16     cputch('\n', &cnt);
17     return cnt;
18 }
```

代码 5-10: `vprintk` 改造代码

至此，我们完成了串口实现和格式化输出。通过在 K210 上验证，我们的 `sprint` 可以通过串口，在控制台上输出格式化的字符串。

文件系统相关接口

Device Tree 相关接口

PKE 运行在 S 态，设备树文件已经由 M 态的 RustSBI 读取并处理。因此 PKE 不需要读取和解析设备树文件。所以，我们在移植时，只需要去除 PKE 读取 DTB 的代码即可。

shutdown 与 poweroff 接口

内核在编码调试过程中,需要借助一些方法来判断变量值是否符合预期,如 `assert` 方法。如果不符合预期需要打印错误信息,并且让内核 `panic`。那么 `panic` 在 `pke` 上是如何实现的呢? 我们可以阅读 `pke` 实现 `panic` 的代码。

```
1  void do_panic(const char *s, ...) {
2      va_list vl;
3      va_start(vl, s);
4      sprint(s, vl);
5      shutdown(-1);
6      va_end(vl);
7  }
8
9  void shutdown(int code) {
10     sprint("System is shutting down with exit code %d.\n", code);
11     frontend_syscall(HTIFSYS_exit, code, 0, 0, 0, 0, 0, 0);
12     while (1)
13         ;
14 }
```

代码 5-11: panic 实现代码

通过观察我们可以发现, `panic` 的调用链路是:

`do_panic`→`shutdown`→`frontend_syscall`

最终 `panic` 是通过 `frontend_syscall` 调用 `spike` 提供的接口实现的。既然是需要调用 `spike` 的 HTIF, 与 `spike` 的 HTIF 具有依赖关系, 那么我们移植的时候就需要去除相关依赖, 自行实现 `do_panic` 函数。

通过观察 `panic` 和 `poweroff` 功能, 很容易发现, 他们都是打印报错信息, 然后终止了硬件线程 (`hart`)。那么我们可以通过调用 SBI 的 `shutdown` 接口来实现这两个接口, 打印报错以后以后, 终止所有的 `hart`。

定时器接口

参考 `ucore` 通过调用 SBI 接口来实现定时器接口。

5.6 用户程序加载

你好

5.6.1 单个进程加载

你好

5.6.2 多进程支持

你好

第 6 章 代理内核实验的参考实现

6.1 系统调用

你好

6.1.1 实验预期

你好

6.1.2 具体实现

6.1.3 执行流程

6.2 异常处理

6.2.1 实验预期

6.2.2 实验分析

6.3 定时器中断

6.3.1 实验预期

6.3.2 实验改进

6.3.3 具体实现

6.3.4 执行流程

6.4 虚拟地址和物理地址的转换

6.4.1 实验预期

6.4.2 具体实现

6.4.3 执行流程

6.5 基本的内存管理

6.5.1 实验预期

6.5.2 具体实现

6.5.3 执行流程

6.6 栈空间不足与缺页异常

你好

6.6.1 实验预期

6.6.2 具体实现

6.6.3 执行流程

6.7 创建子进程 fork 的实现

6.7.1 实验预期

6.7.2 兼容 K210 与改进

6.7.3 具体实现

6.7.4 执行流程

6.8 进程的控制权交接

6.8.1 实验预期

6.8.2 具体实现

6.8.3 执行流程

6.9 进程的时间片调度

6.9.1 实验预期

6.9.2 具体实现

6.9.3 执行流程

6.10 实验指导书的编写及管理

6.10.1 实验指导书的编写

6.10.2 实验指导书的管理

6.10.3 对应代码库的管理

你好

结 论

本文结论……。^[1]

结论作为毕业设计（论文）正文的最后部分单独排写，但不加章号。结论是对整个论文主要结果的总结。在结论中应明确指出本研究的创新点，对其应用前景和社会、经济价值等加以预测和评价，并指出今后进一步在本研究方向进行研究工作的展望与设想。结论部分的撰写应简明扼要，突出创新性。阅后删除此段。

结论正文样式与文章正文相同：宋体、小四；行距：22 磅；间距段前段后均为 0 行。阅后删除此段。

参考文献

参考文献书写规范

参考国家标准《信息与文献参考文献著录规则》【GB/T 7714—2015】，参考文献书写规范如下：

1. 文献类型和标识代码

普通图书：M 会议录：C 汇编：G 报纸：N

期刊：J 学位论文：D 报告：R 标准：S

专利：P 数据库：DB 计算机程序：CP 电子公告：EB

档案：A 舆图：CM 数据集：DS 其他：Z

2. 不同类别文献书写规范要求

期刊

[序号] 主要责任者. 文献题名 [J]. 刊名, 出版年份, 卷号 (期号): 起止页码.

普通图书

[序号] 主要责任者. 文献题名 [M]. 出版地: 出版者, 出版年. 起止页码. [2]

[1] 李成智, 李小宁, 田大山. 飞行之梦: 航空航天发展史概论[M]. 北京: 北京航空航天大学, 2004.

[2] Raymer, Daniel P. Aircraft design: A Conceptual Approach[M]. Reston, Virginia: American Institute of Aeronautics, 1992.

会议论文集

[序号] 析出责任者. 析出题名 [A]. 见 (英文用 In): 主编. 论文集名 [C]. (供选择项: 会议名, 会址, 开会年) 出版地: 出版者, 出版年. 起止页码. [3]

[3] 孙品一. 高校学报编辑工作现代化特征[C]//张为民. 中国高等学校自然科学学报研究会. 科技编辑学论文集 (2). 北京: 北京师范大学出版社, 1998: 10-22.

专著中析出的文献

[序号] 析出责任者. 析出题名 [A]. 见 (英文用 In): 专著责任者. 书名 [M]. 出版地: 出版者, 出版年. 起止页码. [4]

[4] 罗云. 安全科学理论体系的发展及趋势探讨[M]//白春华, 何学秋, 吴宗之. 21 世纪安全科学与技术的发展趋势. 北京: 科学出版社, 2000: 1-5.

学位论文

[序号] 主要责任者. 文献题名 [D]. 保存地: 保存单位, 年份. [5][6]

[5] 张和生. 嵌入式单片机系统设计[D]. 北京: 北京理工大学, 1998.

[6] Sobieski I P. Multidisciplinary Design Using Collaborative Optimization[D]. United States – Cali-

fornia: Stanford University, 1998.

报告

[序号] 主要责任者. 文献题名 [R]. 报告地: 报告会主办单位, 年份. [7][8]

[7] 冯西桥. 核反应堆压力容器的 LBB 分析[R]. 北京: 清华大学核能技术设计研究院, 1997.

[8] Sobieszczanski-Sobieski J. Optimization by Decomposition: A Step from Hierarchic to Non-Hierarchic Systems[R]. NASA CP-3031, 1989.

专利文献

[序号] 专利所有者. 专利题名 [P]. 专利国别: 专利号, 发布日期. [9]

[9] 姜锡洲. 一种温热外敷药制备方案: 88105607[P]. 中国. 1989-07-26.

国际、国家标准

[序号] 标准代号. 标准名称 [S]. 出版地: 出版者, 出版年. [10]

[10] GB/T 16159—1996. 汉语拼音正词法基本规则[S]. 北京: 中国标准出版社, 1996.

报纸文章

[序号] 主要责任者. 文献题名 [N]. 报纸名, 出版年, 月 (日): 版次. [11]

[11] 谢希德. 创造学习的思路[N]. 人民日报, 1998-12-25(10).

电子文献

[序号] 主要责任者. 电子文献题名 [文献类型/载体类型]. 电子文献的出版或可获得地址 (电子文献地址用文字表述), 发表或更新日期/引用日期 (任选). [12]

[12] 姚伯元. 毕业设计 (论文) 规范化管理与培养学生综合素质[EB/OL]. 中国高等教育网教学研究. [2013-03-26]. <http://www.cnnic.net.cn/hlwfzyj/hlwxxzb/201201/P020120709345264469680>.

关于参考文献的未尽事项可参考国家标准《信息与文献参考文献著录规则》(GB/T 7714—2015)

附 录

附录相关内容...

附录 A L^AT_EX 环境的安装

L^AT_EX 环境的安装。

附录 B B_IThesis 使用说明

B_IThesis 使用说明。

附录是毕业设计（论文）主体的补充项目，为了体现整篇文章的完整性，写入正文又可能有损于论文的条理性、逻辑性和精炼性，这些材料可以写入附录段，但对于每一篇文章并不是必须的。附录依次用大写正体英文字母 A、B、C……编序号，如附录 A、附录 B。阅后删除此段。

附录正文样式与文章正文相同：宋体、小四；行距：22 磅；间距段前段后均为 0 行。阅后删除此段。

致 谢

值此论文完成之际，首先向我的导师……

致谢正文样式与文章正文相同：宋体、小四；行距：22 磅；间距段前段后均为 0 行。阅后删除此段。