**《嵌入式系统设计技术》课程设计报告**

|  |  |  |  |
| --- | --- | --- | --- |
| **实验题目** | **(a) 嵌入式操作系统的虚拟化机制设计、(d) 嵌入式操作系统的系统调用编译设计** | | |
| **小组成员** | **李万达、何一鸣、章洮与** | | |
| **实验时间** | **2024.4.27** | **实验地点** | **电子科技大学深圳研究院** |
| **实验成绩** |  | **实验性质** | **R验证性 □设计性 □综合性** |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | |
| **一、实验目的**  本课程设计的主要目的是让学生深入理解和掌握嵌入式系统设计中多核处理器优化、操作系统虚拟化和容错恢复机制。通过本课程设计，：   * 掌握多核处理器嵌入式系统内核设计的关键概念，包括并行处理、任务分配和核间通信。 * 理解并实现嵌入式操作系统的虚拟化技术，能够在单一硬件平台上高效管理和调度多个操作系统实例。 * 学习并应用容错和恢复机制，确保嵌入式系统在遇到不可预测的硬件或软件失败时能够稳定运行或快速恢复。 * 通过实际项目设计和实现，提升分析、评估和优化嵌入式系统设计的能力。 * 提升团队合作、问题解决和创新能力。   **二、实验背景**  (b) 嵌入式操作系统的虚拟化机制设计：  随着嵌入式系统的发展和普及，嵌入式操作系统在各个领域的应用越来越广泛。嵌入式系统通常具有资源受限、功耗低、实时性强等特点，因此对于嵌入式操作系统的虚拟化机制设计提出了更高的要求。虚拟化技术能够有效地提高嵌入式系统的资源利用率、降低系统开发和维护成本，并且能够实现多个应用程序或操作系统的隔离和安全性保护。因此，设计一种适用于嵌入式操作系统的虚拟化机制成为当前的研究热点之一。在这样的背景下，开展嵌入式操作系统的虚拟化机制设计实验具有重要的理论和实践意义。  (d) 嵌入式操作系统的系统调用编译设计：  系统调用是操作系统提供给用户空间程序访问内核功能的接口，是操作系统与用户空间之间的桥梁。在嵌入式系统中，由于资源受限、硬件平台差异等因素，系统调用的设计和优化对系统性能和资源消耗具有重要影响。因此，设计一种高效的系统调用编译方案成为提高嵌入式操作系统性能的关键之一。在这样的背景下，开展嵌入式操作系统的系统调用编译设计实验旨在深入探究系统调用的编译原理和优化策略，为嵌入式系统的性能提升提供技术支持和理论指导。 | | | |
| **三、实验项目内容**  **实验名称：**(b) 嵌入式操作系统的虚拟化机制设计与(d) 嵌入式操作系统的系统调用编译设计  **实验设备：**一台笔记本  **主要功能：**   * **嵌入式操作系统的虚拟化机制设计**：设计并实现一个嵌入式操作系统的虚拟化机制，该机制应支持在单个硬件平台上同时运行多个隔离的操作系统实例。设计目标是创建一个轻量级的虚拟化层（Hypervisor），能够有效地管理和调度多个虚拟机实例，实现资源的隔离和共享，同时保证每个虚拟机实例的高性能和响应性。 * **面向多核处理器的嵌入式操作系统内核设计**：设计并实现一个嵌入式操作系统内核，该内核应支持多核并行处理、任务分配、同步以及高效的核间通信。设计目标是模拟一个嵌入式多核环境，其中内核能够有效地管理和调度多个核上的任务，实现资源共享和任务同步，同时保证系统的高性能和响应性。   **仿真方式：**使用Python编程语言进行编程仿真  **提交内容：①** 实验报告文档（本文档）**②**实验源码**（****②作为附件上传）**  **四、操作指南**  (b)**面向多核处理器的嵌入式操作系统内核设计**：  步骤 1: 实现虚拟机监控器（Hypervisor）   * 初始化Hypervisor： 初始化虚拟机列表。      * 创建虚拟机： 创建并初始化虚拟机实例。      * 启动虚拟机： 更改虚拟机状态并模拟启动过程。     步骤 2：实现虚拟机类  a. 初始化 VirtualMachine：定义 VirtualMachine 类，初始化其 ID，CPU，内存和状态。    b. 启动虚拟机：实现启动方法，改变虚拟机状态为“RUNNING”。    步骤 2：实现虚拟CPU管理   * 定义VirtualCPU类： 包括状态和状态更改方法。      * CPU调度（示例化为轮询调度）： 在Hypervisor中模拟CPU调度。     步骤 3：实现虚拟内存管理   * 定义VirtualMemory类： 包含内存大小和分配方法。   b. 在虚拟机中使用虚拟内存：在VirtualMachine中添加allocate\_memory方法    步骤 4：实现虚拟I/O设备   * 定义VirtualIODevice类： 模拟I/O读写操作。      * 在虚拟机中集成虚拟I/O设备：     步骤5: 实现虚拟机间通信与协作   * 定义VirtualNetworkInterface类： 实现消息队列来模拟网络接口。      * 在虚拟机中使用网络接口来发送和接收消息：       步骤 6：实现简单的虚拟机管理   * 初始化 Hypervisor：创建一个 Hypervisor 实例。      * 创建虚拟机实例：使用 Hypervisor 创建两个虚拟机实例，分别为 vm1 和 vm2。vm1 和 vm2 被分配了不同的资源配置。      * 启动虚拟机：启动两个虚拟机实例。      * 模拟虚拟机间通信：vm1 发送消息给 vm2。vm2 接收来自 vm1 的消息。vm2 应该接收到来自 vm1 的 "Hello VM2" 消息，并打印出来。      * 停止虚拟机：停止两个虚拟机实例。   (d) **面向多核处理器的嵌入式操作系统内核设计**  步骤1 使用VMware安装Linux系统   * 安装VMware，官网可以提供30天的试用或者自行寻找安装包。      * 下载Ubuntu镜像。      * 创建虚拟机，安装Ubuntu。 * 打开 VMware，点击【创建新的虚拟机】。   在这里插入图片描述  在这里插入图片描述   * 选择 “自定义(高级)” , 然后点击【下一步】。   在这里插入图片描述   * 选择【Workstation 15.x】，然后点击【下一步】。   在这里插入图片描述   * 在安装过程中可以设定虚拟机名称和存放位置，然后点击【下一步】。      * 在安装过程中可以根据需求自行设定处理器配置、内存配置以及磁盘大小。 * 完成上述步骤后，虚拟机列表中会出现刚刚新建的虚拟机：   在这里插入图片描述   * 点击【开启此虚拟机】：   在这里插入图片描述  步骤2 构建Linux进程树   * 进程结构定义，利用进程结构体来实现进程的各种信息记录，例如进程号、进程名、进程的兄弟关系等。      * 进程多叉树遍历，由于系统调用的基本功能为遍历Linux进程树。因此需要采用一种合适的遍历方法，考虑到Linux进程树为一颗多叉树，所以用深度搜索中的前向遍历比较合适。preorder\_traversal\_processtree()函数采用前向遍历方法，list\_for\_each为内核封装的链表遍历函数。      * 系统调用主函数，该函数为系统调用的入口，即进行系统调用时，陷入内核后，根据系统调用号会跳转到这个函数执行。preorder\_traversal\_processtree()函数为进程多叉树遍历函数，其中&init\_task为根进程。​copy\_to\_user()可以理解为内核与用户之间的memcpy()，由于内核与用户之间不共享内存，所以运行于内核态的系统调用无法将获取到的进程数据直接传送给用户，只能通过copy\_to\_user()来实现。     步骤3 编写内核模块   * 内核模块初始化函数mymod\_init()，由于内核模块运行在内核态，所以其拥有极高的权限，可以利用该权限，将系统调用临时替换到系统调用表中的空位置，达到快速调试代码，不用重新编译内核的目的。      * insert\_syscall()函数用于修改系统调用表，将系统调用表的指定位置指向自己写的系统调用函数。其中，disable\_wirte\_protect()和enable\_wirte\_protect()函数分别为关闭写保护（允许改写系统调用表）和开启写保护。      * remove\_syscall()函数用于移除系统调用的过程。     步骤4 系统调用方法的程序测试   * 树状图构建程序编写，构建出一个树状图。利用系统调用传回的数组，构建出Linux进程树图，表现出进程之间的父子兄弟关系。 * 测试进程构建程序编写，运行该程序可以产生一个设计好的进程树。 * 内核模块构建程序编写，内核模块的构建程序形式较为固定。      * 程序编译并运行。   编译测试进程生成程序(setprocess.c)    编译树状图生成程序(mymod\_test.c)    编译内核模块(系统调用)程序(mymod.c)，虽然没有显式地定义并生成mymod.o，但是make有强大的自动推导功能，它可以自动推导出所需依赖    步骤5 系统调用编译进内核   * 下载对应的Linux内核并解压。 * 添加系统调用号。      * 添加系统调用函数声明(不能放在宏定义里)。      * 编译内核并安装。     **五、实验记录**  （详细描述本实验的主要步骤、关键代码、实验图片等）  (b) **嵌入式操作系统的虚拟化机制设计**：：   * 虚拟机监控器（Hypervisor）实现: 设计一个轻量级的Hypervisor框架，支持加载和运行虚拟机实例。实现硬件资源的抽象化和虚拟化，包括CPU、内存和I/O设备。 * 虚拟CPU管理: 开发虚拟CPU调度器，模拟CPU时间片分配给不同的虚拟机。实现虚拟CPU状态管理，包括创建、运行、挂起和恢复等状态。   实现思路：hypervisor对象初始化时，设定可以使用的CPU资源，并执行CPU调度程序。    CPU调度程序的实现：  为每一个CPU创建一个轮询线程，每隔一个固定的时间将此CPU的占用切换给其他虚拟机。    为每一个核创建一个轮训线程。    **(d) 嵌入式操作系统的系统调用编译设计:**  Linux版本 ubuntu-20.04.6  编译内核版本 5.10.216  更新系统调用表，添加系统调用id，在arch/x86/entry/syscalls/syscall\_64.tbl    添加系统调用函数声明，修改arch/x86/include/asm/syscalls.h    在内核源代码添加函数定义 kernel/sys.c    安装编译所需工具  sudo apt-get install gcc make libncurses5-dev openssl libssl-dev  sudo apt-get install build-essential  sudo apt-get install pkg-config  sudo apt-get install libc6-dev  sudo apt-get install bison  sudo apt-get install flex  sudo apt-get install libelf-dev  make menuconfig 先默认操作 再修改config文件  修改结果如图    编译成功后 安装模块  sudo make modules\_install  sudo make install  修改开机启动项 重新开机 切换内核  **六、****实验调试过程**  （详细描述本实验中所遇问题与解决方案等）  (b)**面向多核处理器的嵌入式操作系统内核设计**：   1. 软件环境配置：  * 配置开发环境，包括合适的编译工具链和模拟器，如QEMU等。 * 下载并配置虚拟机监控器（Hypervisor）的开发框架，例如Xen或KVM。  1. Hypervisor设计与实现：  * 设计轻量级的Hypervisor框架，包括对硬件资源的抽象化和虚拟化。 * 实现Hypervisor的加载和初始化过程，确保能够正确识别和管理系统硬件资源。  1. 虚拟CPU管理：  * 开发虚拟CPU调度器，确保能够模拟CPU时间片的分配和调度。 * 实现虚拟CPU状态管理功能，包括虚拟CPU的创建、运行、挂起和恢复等操作。  1. 调试问题与解决方案：  * 遇到问题：在加载和初始化Hypervisor时，出现了无法识别和访问硬件资源的错误。 解决方案：检查Hypervisor的初始化代码，确保正确设置硬件访问权限，并修复硬件识别逻辑，以正确识别系统中的CPU、内存和I/O设备。 * 遇到问题：在实现虚拟CPU调度器时，发现无法正确模拟CPU时间片的分配和调度。 解决方案：检查虚拟CPU调度器的逻辑，确保正确计算和分配CPU时间片，并实现合适的调度算法，如轮转调度或优先级调度，以确保虚拟机能够公平地分享CPU资源。   **(d)面向多核处理器的嵌入式操作系统内核设计**：   1. 系统调用编译设计方案确定：  * 确定实验的系统调用编译设计方案，包括选择适合嵌入式系统的编译器和优化策略。 * 深入研究系统调用在编译阶段的优化技术，如内联函数、尾调用优化等。  1. 系统调用接口设计与实现：  * 设计系统调用的接口规范，包括参数传递方式、调用约定等。 * 在操作系统内核中实现系统调用处理程序，确保能够正确解析用户空间程序的系统调用请求，并执行相应的内核功能。  1. 编译器优化选项配置：  * 配置编译器的优化选项，包括启用内联函数优化、尾调用优化等。 * 调整编译器的优化级别，根据系统需求和性能目标进行适当调整。  1. 调试问题与解决方案：  * 遇到问题：版本不匹配内核编译不通过，解决方案：换用以前版本的内核重新编译。 * 遇到问题：asmlinkage long sys\_mysyscall(void)在高版本的内核中很可能会出现以下报错：   arch/x86/entry/syscall\_64.o:(.rodata+0xa78): undefined reference to '\_\_x64\_sys\_mysyscall'  解决方案：更改为 SYSCALL\_DEFINE0(mysyscall)即可解决。   * 遇到问题：在编译含有系统调用的程序时，出现了链接错误或编译器优化导致的代码逻辑错误。 解决方案：检查系统调用接口规范和调用约定，确保与内核中实现的系统调用处理程序保持一致。同时，调整编译器的优化选项，尝试关闭某些优化策略，以解决代码生成或链接错误。   **七、实验效果展示**  （详细描述本实验中所遇问题与解决方案等）  **(b)嵌入式操作系统的虚拟化机制设计**：  创建三个虚拟机并运行，其中VM1,2占用CPU2，VM3占用CPU3。在运行过程中，设置轮训时间片为5s。    运行结果如下：  VM1,2每隔5s会完成CPU占用的切换，VM3独占CPU，不会完成CPU占用权的切换    设定15s之后VM4开始运行，VM4同样占用CPU3。也就是说在15s之后CPU3由VM3,4共享。    运行结果如下：  在VM4启动之后，CPU调度程序也能完成对CPU3的时间片分配，每隔5s在VM3,4间切换。    创建四个虚拟机，分别输出其真实内存地址。    运行结果如下：    **八、实验总结**  **九、小组分工**  （列出小组组长、成员名字，并详细描述每位所承担工作） | | | |
|  | | | |