Lab-2 系统调用

实验要求

内核的引入

- 磁盘加载，由bootloader加载kernel，由kernel加载用户程序
- 区分内核态与用户态

中断机制

- 完善中断机制与系统中断机制
- 实现具体的系统调用与库函数
- 用户态执行一个系统调用，会经历什么过程？

实验内容

中断机制

为实现用户态服务和保护特权级代码，需要完善以下关键结构：

IDT（中断描述符表）：

- 定义中断向量与处理程序的映射关系。
- 支持中断门、陷阱门和任务门。

TSS（任务状态段）：

- 存储任务的状态信息，包括堆栈指针。
- 用于特权级切换时的堆栈管理。

中断处理程序：

- 处理硬件中断、异常和软中断。
- 保存和恢复上下文，确保程序正确执行。

实验内容

系统中断处理例程

- 完善系统调用处理函数 sysWrite ， 类似Linux中的系统调用处理函数 _write
- printf()拥有完整的格式化输出，包括%d, %x, %s, %c四种格式转换输出功能。

思考：

- 该如何描述系统中断的处理过程
- 为什么要有系统中断的机制？

实验内容

一些库函数的实现

- 键盘按键的串口回显
- 实现库函数 getChar、getStr 和对应的处理例程
- 增加一个阻塞方式的`sleep` 库函数

增加now系统调用以及对应的库函数实现

- 应用程序每隔1秒钟输出当前的时间

实验提示

实验流程

1. 由实模式开启保护模式并跳转到bootloader
2. 由bootloader加载kernel
3. 完善kernel相关的初始化设置
4. 由kernel加载用户程序
5. 实现用户需要的库函数
6. 用户程序调用自定义实现的库函数完成格式化输入输出，通过测试代码

建议：

- 运行 QEMU 时若显示异常但代码逻辑无误，可能是环境配置问题
- 提问前请明确自己的疑问，并结合自身的思考，不要笼统地询问“怎么做”或“哪里错了”

相关资料

串口输出

串口初始化与输出

- 在 lab2/kernel/main.c 中，initSerial() 用于初始化串口设备。
- 通过 putChar()（定义在 serial.h，实现在 serial.c）可以向串口输出字符。

调试接口 assert()

- 框架代码基于 putChar() 提供了 assert() 接口（定义在 assert.h），方便调试时使用。

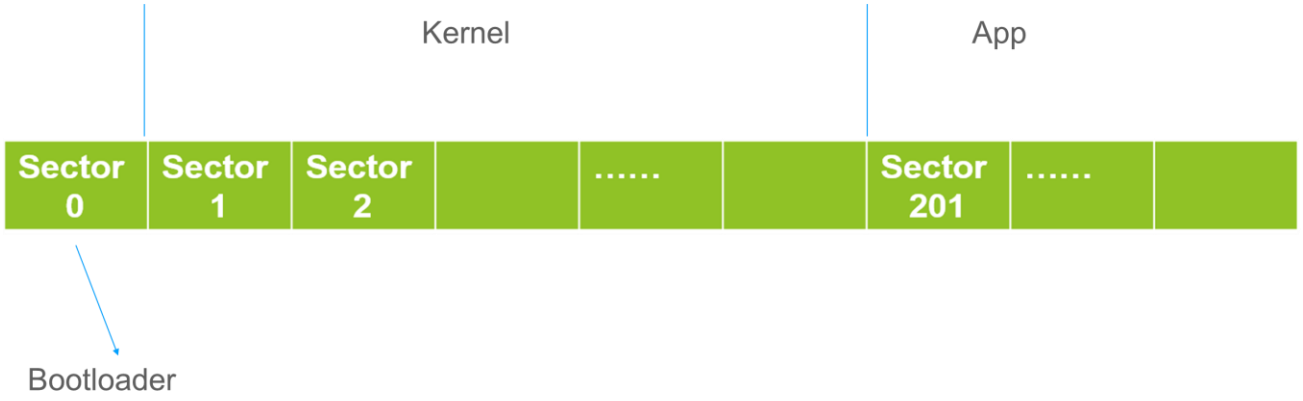
QEMU 串口输出

- 在 Makefile 中，-serial stdio 选项将 QEMU 模拟的串口数据输出到宿主机的标准输出。

相关资料

从系统启动到用户程序

1. 初始化硬件设备
2. 初始化核心数据结构
3. 加载用户程序
4. 调用库函数



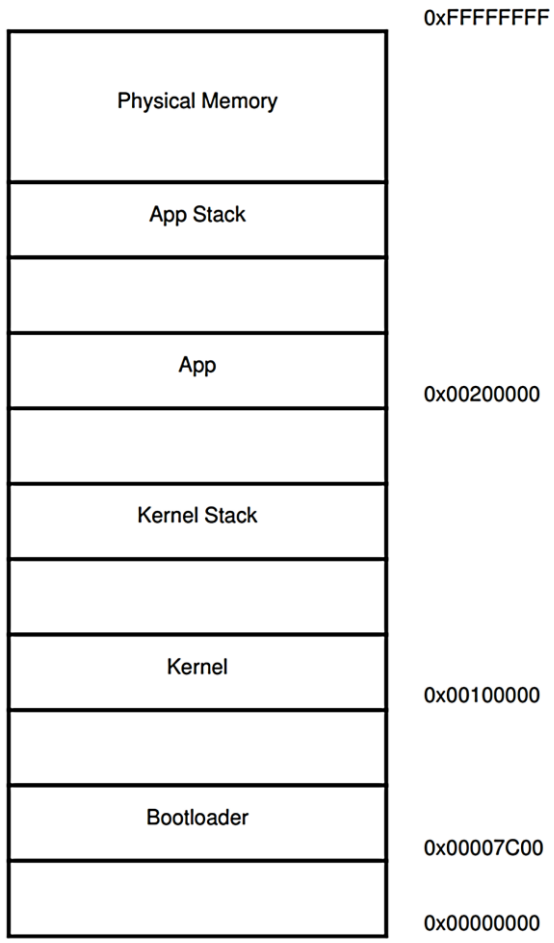
加载程序-内存分布

内核的内存布局

- 编译时，内核的 .text 段起始地址设为 0x100000。
- GDT 中，内核数据段的基地址设置为 0x0。
- 内核加载到物理内存 0x100000 开始的位置。

用户程序的内存布局

- 编译时，用户程序的 .text 段起始地址设为 0x200000
- GDT 中，用户程序数据段的基地址设置为 0x0。
- 用户程序加载到物理内存 0x200000 开始的位置。



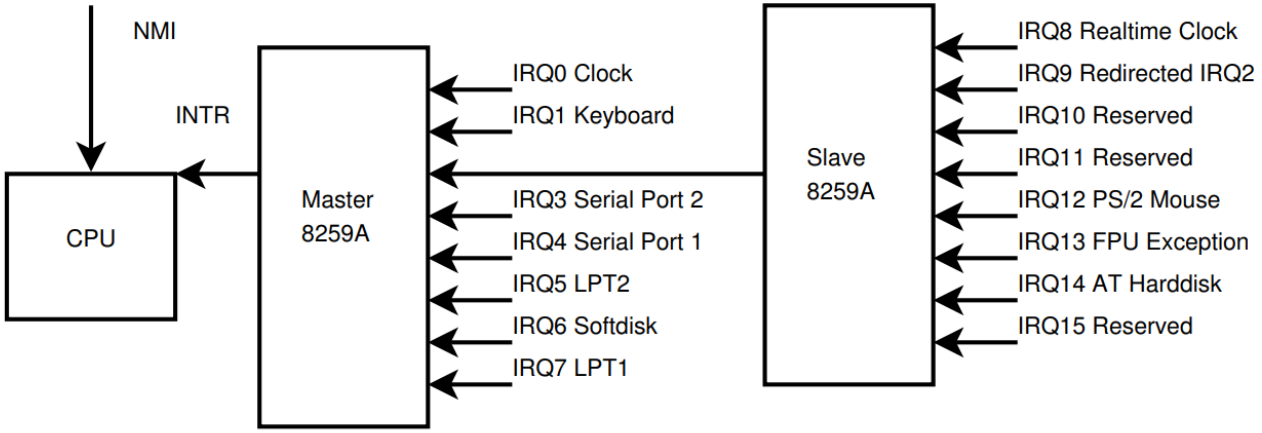
IA-32的中断机制

保护模式下的中断

- 外部硬件中断（Interrupt）
- 异常（Exception）
- 软中断（Software Interrupt）

可屏蔽中断的控制

- CPU控制
- 8259A控制



相关资料

IA-32的中断机制

特点

- 由硬件随机产生，异步于程序执行。
- 可在程序执行的任何时候发生。

可屏蔽中断（Maskable Interrupt）

- 可以被 CPU 屏蔽的中断类型。
- I/O 设备发出的中断请求（IRQ）通过 8259A 可编程中断控制器（PIC）处理，并转化为 8 位中断向量输入 CPU。
- 通过设置 8259A，可对每个 IRQ 分别屏蔽。

非屏蔽中断（Nonmaskable Interrupt, NMI）

- 不能被 CPU 屏蔽的中断类型。
- 由特定危机事件触发，例如硬件故障或掉电。
- 通过 NMI 引脚输入 CPU。

相关资料

IDT

中断向量与 IDT

- 每个中断（异常、硬件中断、软中断）由 8 位中断向量标识，共 256 个中断向量。
- IDT（中断描述符表）包含 256 个表项，每个表项称为门描述符（Gate Descriptor），占 8 字节。

中断处理流程

- 中断发生后，硬件根据中断向量从 IDT 中找到对应的门描述符。
- 结合 GDT（全局描述符表），定位中断处理程序的入口地址并跳转执行。

门描述符的类型

- 任务门（Task Gate）：用于任务切换，现代操作系统中一般不使用。
- 中断门（Interrupt Gate）：执行时禁用中断（IF 位清零），避免嵌套中断。
- 陷阱门（Trap Gate）：执行时不禁用中断，允许嵌套中断。

相关资料

中断处理与状态保存

中断对程序执行的影响

- 中断会打断当前任务 A 的执行，转去处理中断任务 B。
- 处理完成后，CPU 恢复任务 A 的执行，使其“感觉”中断从未发生。

任务 A 的状态表征

- **硬件自动保存：**
  - EIP（指令指针）：指示任务 A 被打断时正在执行的指令。
  - EFLAGS（标志寄存器）：保存 CPU 的状态标志。
  - CS（代码段寄存器）：包含当前代码段的段选择子和特权级（CPL）。
- **手动保存：**
  - 通用寄存器（GPR）：需在中断处理程序中手动保存和恢复。

相关资料

中断处理流程

确定中断向量

- 根据中断或异常确定向量号 i（0-255）。

查找门描述符

- 读取 IDTR 寄存器指向的 IDT 中的第 i 项门描述符。
- 从 GDTR 寄存器获取 GDT 基地址，查找段选择子对应的段描述符。

特权级检查

- 比较 CS 寄存器的 CPL 和门描述符的 DPL，确保中断处理程序的权限不低于引起中断的程序。

堆栈切换（特权级变化时）

- 读取 TR 寄存器，访问当前进程的 TSS。
- 从 TSS 中加载与新特权级相关的 SS 和 ESP。
- 在新堆栈中保存旧堆栈的 SS 和 ESP。

相关资料

IRET 指令

功能

- 用于从中断处理程序返回到被中断的程序。
- 从高特权级返回低特权级。

操作

- 从栈顶依次弹出 EIP、CS 和 EFLAGS，恢复被中断程序的执行状态。
- 若弹出的 CS 的 CPL 大于当前 CPL，继续弹出 ESP 和 SS，切换堆栈。

相关资料

系统调用

系统调用的本质

- 系统调用是基于中断机制封装的一层用户接口。
- 用户程序通过软中断（如 int 0x80）触发系统调用，切换到内核态执行特权操作。
- 内核完成操作后，返回用户态并恢复用户程序的执行。

系统调用的作用

- 提供安全的用户接口，保护内核代码和数据。
- 实现用户程序与硬件资源的隔离。
- 支持多任务和资源共享。

系统调用的参数传递

系统调用号

- 每个系统调用至少需要一个参数，即系统调用号，用于指示执行哪个内核函数。

参数传递方式

- 普通 C 语言函数通过堆栈传递参数（从右向左压栈）。
- 系统调用涉及用户态到内核态的堆栈切换，无法直接使用堆栈传递参数。
- 框架代码通过 eax、ebx 等通用寄存器传递参数。

内核态参数处理

- 在 kernel/irqHandle.c 中，使用 TrapFrame 数据结构保存内核堆栈中的寄存器值。
- 通用寄存器的值通过 pushal 指令压入内核堆栈。

相关资料

RTC获取日期信息

RTC I/O 端口

- 0x70：选择寄存器。
- 0x71：数据寄存器。

CMOS 内存

- 存储 BIOS 设置和 RTC 数据。
- 非易失性内存。

时间数据结构

- 秒、分、时、日、月、年分别存储在特定寄存器中。

获取时间

- 读取 RTC 寄存器。
- 将 BCD 格式转换为数字格式。

解决思路

键盘按键的串口回显

键盘中断触发

- 用户按下或释放按键时，键盘接口接收到扫描码。
- 键盘接口触发中断请求（IRQ）。

中断服务程序

- 键盘中断服务程序从接口获取扫描码。
- 根据扫描码判断用户按下的键并进行处理。

中断处理完成

- 通知中断控制器中断已处理完毕。
- 执行中断返回（IRET）。

解决思路

设置门描述符

设置键盘中断门描述符

- 在 IDT 中添加键盘中断的描述符，键盘中断号为 0x21，对应 IRQ1。

初始化 IDT

- 在 initIdt() 中配置中断描述符，映射到处理函数 irqKeyboard()。

硬件中断

- 硬件中断不受 DPL 影响，8259A 的中断为内核级，无法被用户程序模拟。

按键处理

- 每次按键触发时，内核调用 irqKeyboard() 进行处理。

解决思路

完善中断服务例程

中断触发到 keyboardHandle()

当键盘中断触发时，执行流程最终会落到 keyboardHandle() 函数。

实现串口回显

在 keyboardHandle() 中，使用键盘驱动接口和串口输出接口来完成键盘按键的串口回显。这样，按下的每个键都会显示在 stdio 上。

显示方式选择

除了通过串口回显外，还可以通过直接操作显存，将按键打印到屏幕上，提供更加直观的输出。

解决思路

在保护模式下实现定时器

定时器初始化

设置 8253 定时器产生时钟中断，定时器中断的实际中断号为 0x20。

中断号与 IDT 设置

中断号 0x20 用于定时器中断，需在 IDT 中绑定该中断号与对应的处理程序。

中断处理

定时器中断会触发中断处理程序，已在 kernel/doIrq.S 中提供，配置 IDT 后即可响应定时器中断。

解决思路

设置sleep()库函数

定时器初始化

设置 8253 定时器产生时钟中断，定时器中断的实际中断号为 0x20。

中断号与 IDT 设置

中断号 0x20 用于定时器中断，需在 IDT 中绑定该中断号与对应的处理程序。

中断处理

定时器中断会触发中断处理程序，已在 kernel/doIrq.S 中提供，配置 IDT 后即可响应定时器中断。

解决思路

实现 printf() 的处理例程

系统调用设置

- 系统调用中断号设为 0x80，处理函数为 irqSyscall()。
- 门描述符的 DPL 设置为用户级，允许用户程序触发中断。

系统调用流程

- 用户程序通过 int $0x80 触发系统调用。
- 内核根据系统调用号选择处理例程。
- 显存操作在 sysPrint() 函数中完成

解决思路

实现 now()库函数

now()的描述

- 返回当前的系统时间

展示now()的功能

- 在测试程序中，每隔1000ms，调用now()，打印当前时间

实验提交

实验提交截至日期：4月1日 23:55 +08:00

提交注意事项：同Introduction