OS LAB 2 系统调用

个人信息

姓名：郑凯琳

学号：205220025

邮箱：205220025@smail.nju.edu.cn

1. 实验目的

（1）程序加载：bootloader 加载 kernel，kernel 加载用户程序

（2）完善中断机制：在 kernel 部分完善中断机制，为用户系统调用提供服务

（区分内核态与用户态）

（3）系统调用：在用户程序实现系统调用的实例测试。

2. 实验进度

我完成了所有内容，成功通过磁盘加载引入内核，区分内核态和用户态，完善中断机制，并通过实现用户态 I/O 函数基于中断实现系统调用的全过程。

3. 实验过程

（1）从实模式进入保护模式，加载内核至内存，执行跳转

- bootmain 函数：填写函数入口和偏移量来加载内核程序。

readSect 函数已从磁盘中读出引导程序所处的扇区。

- kMainEntry：将 elf 地址转换为 struct ELFHeader 类型的指针，通过指针访问 ELF 头部的成员，得程序的入口点地址。
- phoff：通过 elf 地址加上 ELF 头部的程序头表偏移量 phoff，得到程序头表的地址，
- offset：再从程序头表中获取第一个段的偏移量 offset，将其赋值给 offset。

```
// TODO: 阅读boot.h查看elf相关信息，填写kMainEntry、phoff、offset
// 将 elf 地址转换为 struct ELFHeader 类型的指针
kMainEntry = (void(*)(void))((struct ELFHeader *)elf)->entry; // 访问 ELF 头部的成员，获取程序的入口点地址
// 程序头表的偏移量
phoff = ((struct ELFHeader *)elf)->phoff;
// .text 段在内存中的偏移量
offset = ((struct ProgramHeader *)(elf + phoff))->off;
```

bootloader.c

（2）完善 kernel 相关的初始化设置

- 初始化中断门和陷阱门：这些门是中断描述符表（IDT）中的条目，用于管理处理器响应中断和异常时的行为。

setIntr 函数和 setTrap 函数根据给定的参数设置中断门和陷阱门的属性，并将这些门插入到中断描述符表中。通过调用这些函数，可以为不同类型的中断和异常设置相应的处理程序，并指定它们的特权级别。

```
/* 初始化一个中断门(interrupt gate) */
static void setIntr(struct GateDescriptor *ptr, uint32_t selector, uint32_t offset, uint32_t dpl) {
        // TODO: 初始化interrupt gate
        ptr->offset_15_0 = offset & 0xFFFF; // 中断服务程序的低位偏移量
        ptr->segment = KSEL(selector); // 段选择子
        ptr->pad0 = 0; // 未使用的填充字段
        ptr->type = INTERRUPT_GATE_32;
        ptr->system = 0; // false // 是一个中断门而不是系统调用门
        ptr->privilege_level = dpl; // 给定的特权级别
        ptr->present = 1; // true // 门是存在的
        ptr->offset_31_16 = (offset >> 16) & 0xFFFF; // 中断服务程序的高位偏移量
}
```

idt.c

- 初始化 IDT 表，为中断设置中断处理函数：

首先在 doIrq.S 汇编代码中补全 keyboard 事件 的中断事件码 0x21，

再根据给出的函数，在 idt.c 中为各中断补全中断处理函数。

- 补充内核 main 函数：

各 init 函数已实现好，只需要加入各 init 函数即可。

- 填充中断处理程序的调用（irqHandle 函数）：

根据各中断给出的中断号，在 irqHandle.c 文件中，根据不同的中断填充其对应调用的处理函数。

（3）由 kernel 加载用户程序（完善中断机制）

填写中断处理函数：填写未完全实现的 keyboardHandle，syscallPrint，syscallGetChar，syscallGetStr 等函数。（irqHandle.c 文件）具体可使用 keyBuffer 数组来辅助相应功能的实现。

- keyboardHandle 函数：

TODO —— 处理正常的字符

1. 当接收到的键盘扫描码小于 0x81 时，表示接收到的是正常的可显示字符，而不是特殊按键（例如功能键、控制键等）。

2. 首先通过 getChar 函数将键盘扫描码转换为字符，并检查该字符是否可显示（ASCII 码大于等于 0x20）。

3. 如果是可显示字符，则将其打印到屏幕上，并将字符存储到键盘缓冲区中。

4. 然后，将字符及其颜色属性（前景色为亮红色）写入到显示缓冲区中对应位置，以便在屏幕上显示。

5. 同时，更新显示的列和行，如果显示列超过了屏幕宽度，重置显示列并增加显示行；如果显示行超过了屏幕高度，则执行屏幕滚动并更新显示行和列。

```
displayCol++;
if (displayCol >= 80) // 显示列超过屏幕宽度
{
        displayCol = 0;
        displayRow++;
}
while (displayRow >= 25) // 显示行超过屏幕高度
{
        scrollScreen();
        displayRow--;
        displayCol = 0;
}
```

- syscallPrint 函数：系统调用函数，用于在屏幕上打印字符串。

TODO —— 完成光标的维护和打印到显存

1. 遍历字符串中的每个字符，判断是否为换行符。

2. 如果是换行符，则将显示列归零，并增加显示行数；否则，将字符与颜色属性合并为显示数据，然后写入显示缓冲区中对应位置，并更新显示列。

3. 如果显示列达到了屏幕宽度，将显示列归零并增加显示行数。

4. 如果显示行数超过了屏幕高度，执行屏幕滚动，确保显示行数不超过屏幕高度。

- syscallGetChar 函数：从键盘缓冲区中获取一个字符，

并根据换行符的存在与否，决定是否将其移除，并将下一个字符作为返回值。

TODO —— 实现

1. 检查键盘缓冲区中最后一个字符是否为换行符（'\n'）：如果是换行符，则将 `hasNewline` 标志设置为 1，并将缓冲区中的换行符移除。

2. 返回下一个字符：如果缓冲区中仍有字符，并且 hasNewline 标志为 1（即之前存在换行符），则将下一个字符作为返回值，并更新缓冲区的指针，指向下一个字符。

3. 处理缓冲区为空的情况：如果缓冲区为空，或者缓冲区中没有换行符，那么将返回值设为 0，表示没有可用的字符。

- syscallGetStr 函数：从键盘缓冲区中获取一个字符串，

并将其复制到指定的内存地址中。在复制过程中，根据换行符的存在与否和目标字符串的长度，决定是否进行复制操作，并将相应的结果返回给调用者。

TODO —— 实现

1. 初始化变量和寄存器：初始化 hasNewline 为 0，用于标记缓冲区中最后一个字符是否为换行符。同时，初始化变量 i 为 0，用于循环计数；设置了 sel 为用户数据段的选择子，并将其存入 es 寄存器中。

2. 检查键盘缓冲区中最后一个字符是否为换行符（'\n'）：如果是换行符，则将 hasNewline 标志设置为 1，并将缓冲区中的换行符移除。

3. 复制字符串到目标地址：如果缓冲区中存在换行符（即 hasNewline 为 1），或者缓冲区中的字符数量大于等于要求的字符串长度（tf->ebx），则将缓冲区中的字符串复制到目标地址（tf->edx）中。复制过程中，先打印字符串的长度，然后使用内联汇编将缓冲区中的字符逐个复制到目标地址中。

4. 返回值处理：如果成功复制了字符串，则将返回值 tf->eax 设为 1；否则，将返回值设为 0，表示缓冲区中的字符数量不足，无法复制字符串。

```
for (i = 0; i < min(tf->ebx, bufferTail - bufferHead); i++)
{
        asm volatile("movb %1, %%es:(%0)"::"r"(tf->edx + i), "r"(keyBuffer[bufferHead + i]));
}
tf->eax = 1; // Successfully copied the string
```

（4）实现用户需要的库函数

在 app/main.c 中调用了 printf 和 scanf，下面分别阐明调用过程和实现机制（lib/syscall.c）。

- printf 函数：根据给定的格式字符串和参数，将格式化的文本输出到标准输出。

1. 遍历格式字符串中的每个字符。

2. 如果遇到 %，则表示后面是一个格式化参数。

3. 根据格式化参数的类型（%d %x %s %c），从参数列表中获取相应的值，并将其转换为字符串。

4. 将转换后的字符串添加到输出缓冲区中。

- getChar 函数：

判断返回值是否为 0 来进行返回。

```
char getChar(){ // 对应SYS_READ STD_IN
        // TODO: 实现getChar函数，方式不限
        char ret = 0; // 初始化返回值为0，表示暂时没有读取到字符
        // 循环直到成功读取一个字符
        while (ret == 0)
        {
                ret = (char)syscall(SYS_READ, STD_IN, 0, 0, 0, 0); // 调用系统调用读取标准输入的字符，返回值存储在 ret 中
        }
        return ret; // 返回读取到的字符
}
```

- getStr 函数：

同样地，判断返回值是否为 0 来进行返回。

```
void getStr(char *str, int size){ // 对应SYS_READ STD_STR
        // TODO: 实现getStr函数，方式不限
        int ret = 0;
        while (ret == 0)
        {
                ret = syscall(SYS_READ, STD_STR, (uint32_t)str, size, 0, 0); // 调用系统调用读取字符串，将结果存储在 str 中，最多读取 size 个字符
        }
        return;
}
```

4. 实验结果

（1）用户程序测试

```
I/O test begin...
the answer should be:
##############################################################
Hello, welcome to OSlab! I'm the body of the game.
Now I will test your printf:
1 + 1 = 2, 123 * 456 = 56088, 0, -1, -2147483648, -1412505855, -32768, 102030, 0
, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
Now I will test your getChar: 1 + 1 = 2
2 * 123 = 246
Now I will test your getStr: Alice is stronger than Bob
Bob is weaker than Alice
##############################################################
your answer:
==============================================================
Hello, welcome to OSlab! I'm the body of the game.
Now I will test your printf:
1 + 1 = 2, 123 * 456 = 56088, 0, -1, -2147483648, -1412505855, -32768, 102030, 0
, ffffffff, 80000000, abcedf01, ffff8000, 18e8e
Now I will test your getChar: 1 + 1 = 2
2 * 123 = 246
Now I will test your getStr: Alice is stronger than Bob
Bob is stronger than Alice
==============================================================
Test end!!! Good luck!!!
```

（2）可以输入字符，可以实现退格

```
Test end!!! Good luck!!!
abcde
abcd_
```