Operating system

Lab 6

Author: 吴杭

Date: 2024/12/24

Autumn-Fall 2024-2025 Semester

Table of Contents

Chapter 1: 实验流程

1.1) 实验步骤

准备工作按照实验文档上的要求一致，这里不展开。

1.1.1) 4.2 shell 与内核进行交互

首先根据实验文档中的指引，修改 `proc.h` 中 `task_struct` 的结构，增加指向文件表的指针。

1.1.1.1) 4.2.2 `stdout/err/in` 初始化

完善 `file_init`，我们只需要按照实验文档提供的示例来完成即可。首先根据 `struct file_struct` 的大小来 alloc 相应的内存。然后设置好 `stdou/err/in` 的各个成员变量即可。

```
struct files_struct *file_init() {
    // todo: alloc pages for files_struct, and initialize stdin, stdout, stderr
    // alloc pages for file struct
    struct files_struct *ret = NULL;
    // calculate the size of files_struct and the pages number it needs
    uint64_t files_struct_size = sizeof(struct files_struct);
    uint64_t pages = (files_struct_size + PGSIZE - 1) / PGSIZE;
    struct files_struct* files_struct_space = (struct files_struct*)alloc_pages(pages);
    if (files_struct_space == NULL) {
        printk(RED "Failed to alloc pages for files_struct\n" CLEAR);
        return NULL;
    }
    memset(files_struct_space, 0, files_struct_size);
    // initialize stdin, stdout, stderr
    // stdin
    files_struct_space->fd_array[0].opened = 1;
    files_struct_space->fd_array[0].perms = FILE_READABLE;
    files_struct_space->fd_array[0].cfo = 0;
    files_struct_space->fd_array[0].lseek = NULL;
    files_struct_space->fd_array[0].write = NULL;
    files_struct_space->fd_array[0].read = stdin_read;
    // stdout
    files_struct_space->fd_array[1].opened = 1;
    files_struct_space->fd_array[1].perms = FILE_WRITABLE;
    files_struct_space->fd_array[1].cfo = 0;
    files_struct_space->fd_array[1].lseek = NULL;
    files_struct_space->fd_array[1].write = stdout_write;
    files_struct_space->fd_array[1].read = NULL;
    // stderr
    files_struct_space->fd_array[2].opened = 1;
    files_struct_space->fd_array[2].perms = FILE_WRITABLE;
    files_struct_space->fd_array[2].cfo = 0;
    files_struct_space->fd_array[2].lseek = NULL;
    files_struct_space->fd_array[2].write = stderr_write;
    files_struct_space->fd_array[2].read = NULL;
    // ensure other files are not opened
    for (int i = 3; i < MAX_FILE_NUMBER; i++) {
        files_struct_space->fd_array[i].opened = 0;
    }
```

```
    ret = files_struct_space;
    return ret;
}
```

1.1.1.2) 4.2.3 处理 stdout/err 的写入

sys_write 的逻辑是，首先检查文件是否打开，如果打开了，再检查他的权限是否是 writable，如果是的话就调用 file 中的 write 函数进行写操作。

```
int64_t sys_write(uint64_t fd, const char* buf, uint64_t len){
    int64_t ret;
    struct file *file = &(current->files->fd_array[fd]);
    if (file->opened == 0){
        printk("file not opened\n");
        return -1;
    } else {
        // check perm and call write function of file
        // check perm
        if (!(file->perms & FILE_WRITABLE)) {
            Err("write into a file that is not writable");
        }
        // call write function of file
        ret = file->write(file, buf, len);
    }
    return ret;
}
```

完善 stderr_write，直接根据 stdout_write 的示例完成即可，但是把输出换成红色的来表示为 err。具体的逻辑就是遍历需要打印的每一个字符，调用 printk 进行输出即可。

```
int64_t stderr_write(struct file *file, const void *buf, uint64_t len) {
    // todo
    char to_print[len + 1];
    for (int i = 0; i < len; i++) {
        to_print[i] = ((const char *)buf)[i];
    }
    to_print[len] = 0;
    return printk(RED "%s" CLEAR, to_print);
}
```

1.1.1.3) 4.2.4 处理 stdin 的读取

实现 sbi_debug_console，其逻辑和其他的 sbi 函数基本一致，设置好 eid 和 fid 之后，把其它参数依次传入即可。

```
struct sbiret sbi_debug_console_read(uint64_t num_bytes, uint64_t base_addr_lo,
uint64_t base_addr_hi) {
    // set eid and fid
    uint64_t sbi_debug_console_read_eid = 0x4442434e;
    uint64_t sbi_debug_console_read_fid = 1;
   return sbi_ecall(sbi_debug_console_read_eid, sbi_debug_console_read_fid, num_bytes,
base_addr_lo, base_addr_hi, 0, 0, 0);
}
```

完善 stdin_read，只需要调用 uart_getchar 来读取 len 个字符即可。

```
int64_t stdin_read(struct file *file, void *buf, uint64_t len) {
    // todo: use uart_getchar() to get `len` chars
    for (int i = 0; i < len; i++) {
        ((char *)buf)[i] = uart_getchar();
    }
}
```

最后在 trap.c 中捕获对应的系统调用进行处理即可。

```
else if(regs->general_regs[16] == SYS_READ) {
      regs->general_regs[9] =  sys_read(regs->general_regs[9], (char *)regs-
>general_regs[10],
                regs->general_regs[11]);
}
```

```
int64_t sys_read(uint64_t fd, char* buf, uint64_t len){
    int64_t ret;
    struct file *file = &(current->files->fd_array[fd]);
    if (file->opened == 0){
        printk("file not opened\n");
        return -1;
    } else {
        // check perm and call read function of file
        // check perm
        if (!(file->perms & FILE_READABLE)) {
            Err("read from a file that is not readable");
        }
        // call read function of file
        ret = file->read(file, buf, len);
    }
    return ret;
}
```

Chpater 3: 心得体会

2.1) 遇到的问题

由于我只完成了简单的前半部分，所以其实没遇到什么问题，最主要的问题是有些宏和 string.c 中的函数之前没实现，然后参考之前的实验文档，完成了这部分内容。剩下的大部分内容就是按照实验的指导走就行。

2.2) 心得体会

这次实验让我更好的理解了文件系统，而且前半部分的实验文档写的非常详细，需要写的代码也基本都有示例让我模仿，所以做的很顺利。

Declaration

We hereby declare that all the work done in this lab 5 is of our independent effort.