浙江大学

本科实验报告

课程名称：	操作系统
姓　名：	徐文皓
学　院：	计算机科学与技术学院
系：	软件工程系
专　业：	软件工程
学　号：	3210102377
指导教师：	夏莹杰

2024 年　01 月　12 日

浙江大学操作系统实验报告

实验名称：　　VFS & FAT32 文件系统

电子邮件地址：__手机：__.

实验地点：　　线上　　实验日期：2024 年 01 月 12 日

一、实验目的和要求

本实验的要求是：为用户态的 Shell 提供 read 和 write syscall 的实现。

二、实验过程

本次实验基于 Lab4 完成。

按照要求调整目录结构，向 include/types.h 中补充一些类型别名。

```
typedef unsigned long uint64_t;
typedef long int64_t;
typedef unsigned int uint32_t;
typedef int int32_t;
typedef unsigned short uint16_t;
typedef short int16_t;
typedef uint64_t* pagetable_t;
typedef char int8_t;
typedef unsigned char uint8_t;
typedef uint64_t size_t;
```

修改 Makefile 文件使得正常编译运行。

修改 task_struct 结构体，新增 files 成员。

```
// arch/riscv/include/proc.h
struct task_struct {
    uint64 state;    // 线程状态
    uint64 counter;  // 运行剩余时间
    uint64 priority; // 运行优先级 1最低 10最高
    uint64 pid;      // 线程id

    struct thread_struct thread;
    pagetable_t pgd;
    struct file *files;
};
```

并在 task_init()中调用 file_init()对其初始化。

```
// arch/riscv/kernel/proc.c
void task_init() {
  ...
  for (int i = 1; i < NR_TASKS; i++) {
    ...
    task[i]->pgd = (pagetable_t)kalloc();
    memcpy(task[i]->pgd, swapper_pg_dir, PGSIZE);

    load_program(task[i]);
    task[1]->files = file_init();
    ...

```

完善 file_init()。这里主要修改了 write 这一函数指针，其余可参考上文。

```c
// fs/vfs.c
struct file* file_init() {
    struct file *ret = (struct file*)alloc_page();

    // stdin
    ret[0].opened = 1;
    ret[0].perms = FILE_READABLE;
    ret[0].cfo = 0;
    ret[0].lseek = NULL;
    ret[0].write = NULL;
    ret[0].read = stdin_read;
    memcpy(ret[0].path, "stdin", 6);
    // ...

    // stdout
    ret[1].opened = 1;
    ret[1].perms = FILE_WRITABLE;
    ret[1].cfo = 0;
    ret[1].lseek = NULL;
    ret[1].write = stdout_write/* todo */;
    ret[1].read = NULL;
    memcpy(ret[1].path, "stdout", 7);

    // stderr
    ret[2].opened = 1;
    ret[2].perms = FILE_WRITABLE;
    ret[2].cfo = 0;
    ret[2].lseek = NULL;
    ret[2].write = stderr_write;
    ret[2].read = NULL;
    memcpy(ret[2].path, "stderr", 7);
    // ...

    return ret;
}
```

补充系统调用处理函数 syscall()，增加对 sys_write()和 sys_read()的调用。

```c
// arch/riscv/kernel/trap.c
void syscall(struct pt_regs* regs) {
    uint64 a7 = regs->x[17];

    if (a7 == SYS_GETPID) {
        regs->x[10] = current->pid;
    }
    else if (a7 == SYS_WRITE) {
        regs->x[10] = sys_write(regs->x[10], (const char*)(regs->x[11]), regs->x[12]);
    }
    else if (a7 == SYS_READ) {
        regs->x[10] = sys_read(regs->x[10], (const char*)(regs->x[11]), regs->x[12]);
    }
    else {
        printk("[S] Unhandled syscall: 0x%lx", a7);
        while (1);
    }
    return;
}
```

其中 sys_write()和 sys_read()的实现如下。

```c
// arch/riscv/kernel/trap.c
uint64 sys_write(unsigned int fd, const char* buf, size_t count) {
    struct file* target_file = &(current->files[fd]);
    if (target_file->opened) {
        return target_file->write(target_file, buf, count);
    } else {
        printk("file not open\n");
        return ERROR_FILE_NOT_OPEN;
    }
}

uint64 sys_read(unsigned int fd, const char *buf, size_t count){
    struct file* target_file = &(current->files[fd]);
    if (target_file->opened) {
        return target_file->read(target_file, buf, count);
    } else {
        printk("file not open\n");
        return ERROR_FILE_NOT_OPEN;
    }
}
```

实现 stdin_read()。

```c
// fs/vfs.c
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();
    }
}
```

stderr_write()和 stdout_write()本质上相同，参照实现即可。

```c
// fs/vfs.c
int64_t stdout_write(struct file* file, const void* buf, uint64_t len) {
    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(buf);
}

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(buf);
}
```

至此，内核可以按照预期运行，允许使用 echo 命令。

```
SWITCH TO [pid=1 counter=4 priority=37]
hello, stdout!
hello, stderr!
SHELL > echo "Goodbye, Operating System!"
Goodbye, Operating System!
SHELL > echo "Thank you all for your delication!"
Thank you all for your delication!
SHELL >
```

三、讨论和心得

本次实验多是模仿已经写好的函数，总体难度很小。相比之下，可能还是配置 Makefile 和文件结构使得在添加相关内容后能够正常运行更加难一些。

至此，操作系统的所有实验都已结束。可以说操作系统是我目前最为“尊敬”的一门课程，自学习这门课起我才觉得自己算是个计算机专业领域的门徒。操作系统的理论实验结合情况可以说在我上过的课程中算得上是很不错的，直到期末复习时也从对实验的回忆中深刻了对理论内涵的理解。

回头看，从处理时钟中断开始，我学习了 GDB、Makefile 等种种有用的工具，也初步地了解了许多操作系统组成部件的实现原理——其中令我印象深刻的还是当看懂 vma_struct、迫不及待地向室友分享时的喜悦——您可以参照我前期报告中出现的那幅图象。当然，困难也是有的，如交叉编译链的使用和页表映射确实花费了我不少的精力，甚至那个思考题让我受到了不小的打击。

但是我们走到了最后。期待有机会再次相见，也期待我们能以此有朝一日建设出伟大的工程。感谢夏莹杰老师和各位助教一路相伴。