任务目标

7. `fcntl` 支持 `F_SETFL` 与 `F_GETFL`：支持修改 file 的 status 和 mode

通过F_GETFL获取文件描述符的status，通过F_SETFL修改文件描述符的status和mode

相关代码

crates > axfs > src > api > port.rs > ...

```rust
bitflags! {
    /// 指定文件打开时的权限
    #[derive(Clone, Copy, Default, Debug)]
    pub struct OpenFlags: u32 {
        /// 只读
        const RDONLY = 0;
        /// 只能写入
        const WRONLY = 1 << 0;
        /// 读写
        const RDWR = 1 << 1;
        /// 如文件不存在，可创建它
        const CREATE = 1 << 6;
        /// 确认一定是创建文件。如文件已存在，返回 EEXIST。
        const EXCLUSIVE = 1 << 7;
        /// 使打开的文件不会成为该进程的控制终端。目前没有终端设置，不处理
        const NOCTTY = 1 << 8;
        /// 同上，在不同的库中可能会用到这个或者上一个
        const TRUNC = 1 << 9;
        /// 非阻塞读写?(虽然不知道为什么但 date.lua 也要)
        /// 在 socket 中使用得较多
        const NON_BLOCK = 1 << 11;
        /// 要求把 CR-LF 都换成 LF
        const TEXT = 1 << 14;
        /// 和上面不同，要求输入输出都不进行这个翻译
        const BINARY = 1 << 15;
        /// 对这个文件的输出需符合 IO 同步一致性。可以理解为随时 fsync
        const DSYNC = 1 << 16;
        /// 如果是符号链接，不跟随符号链接去寻找文件，而是针对连接本身
        const NOFOLLOW = 1 << 17;
        /// 在 exec 时需关闭
        const CLOEXEC = 1 << 19;
        /// 是否是目录
        const DIR = 1 << 21;
    }
} bitflags!
```

修改代码

crates/linux_syscall_api/src/syscall_fs/imp/ctl.rs

```
/// # Arguments
/// * `fd`: usize
/// * `cmd`: usize
/// * `arg`: usize
```

```
pub fn syscall_fcntl64(args: [usize; 6]) -> SyscallResult {
    let fd = args[0];
    let cmd = args[1];
    let arg = args[2];
    let process = current_process();
    let mut fd_table = process.fd_manager.fd_table.lock();

    if fd >= fd_table.len() {
        debug!("fd {} is out of range", fd);
        return Err(SyscallError::EBADF);
    }
    if fd_table[fd].is_none() {
        debug!("fd {} is none", fd);
        return Err(SyscallError::EBADF);
    }
    let file = fd_table[fd].clone().unwrap();
    error!("fd: {}, cmd: {}", fd, cmd);
    match Fcntl64Cmd::try_from(cmd) {
        Ok(Fcntl64Cmd::F_DUPFD) => {
            let new_fd = if let Ok(fd) = process.alloc_fd(&mut fd_table) {
                fd
            } else {
                // 文件描述符达到上限了
                return Err(SyscallError::EMFILE);
            };
            fd_table[new_fd] = fd_table[fd].clone();
            Ok(new_fd as isize)
        }
        Ok(Fcntl64Cmd::F_GETFD) => {
            if file.get_status().contains(OpenFlags::CLOEXEC) {
                Ok(1)
            } else {
                Ok(0)
            }
        }
        Ok(Fcntl64Cmd::F_SETFD) => {
            if file.set_close_on_exec((arg & 1) != 0) {
                Ok(0)
            } else {
                error!("file.set_close_on_exec");
                Err(SyscallError::EINVAL)
            }
        }
        Ok(Fcntl64Cmd::F_GETFL) => {
            error!("get file status : {}", file.get_status().bits() as isize);
            Ok(file.get_status().bits() as isize)
        }
        Ok(Fcntl64Cmd::F_SETFL) => {
            if let Some(flags) = OpenFlags::from_bits(arg as u32) {
                let _ = file.set_status(flags);
                Ok(0)
            } else {
                Err(SyscallError::EINVAL)
            }
        }
```

```
        Ok(Fcntl64Cmd::F_DUPFD_CLOEXEC) => {
            let new_fd = if let Ok(fd) = process.alloc_fd(&mut fd_table) {
                fd
            } else {
                // 文件描述符达到上限了
                return Err(SyscallError::EMFILE);
            };

            if file.set_close_on_exec((arg & 1) != 0) {
                fd_table[new_fd] = fd_table[fd].clone();
                Ok(new_fd as isize)
            } else {
                error!("file.F_DUPFD_CLOEXEC");
                Err(SyscallError::EINVAL)
            }
        }
        _ => {
            error!("error fd: {}, cmd: {}", fd, cmd);
            Err(SyscallError::EINVAL)
        }
    }

}
```

测试验证

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

int main() {
    int fd;
    int flags;

    // 打开文件（这里假设是一个已存在的文件）
    fd = open("file.txt", O_RDONLY);
    if (fd == -1) {
        perror("Failed to open file");
        return 1;
    }

    // 获取当前文件描述符的标志
    flags = fcntl(fd, F_GETFL, 0);
    printf("before change, File status flags (hex): %d\n", flags);
    if (flags == -1) {
        perror("fcntl F_GETFL error");
        return 1;
    }

    // 设置文件描述符
    flags = O_WRONLY;
    if (fcntl(fd, F_SETFL, flags) == -1) {
        perror("fcntl F_SETFL error");
```

```
        return 1;
    }

    printf("File descriptor %d is now in non-blocking mode.\n", fd);
    printf("after change, File status flags (hex): %d\n", flags);

    // 再次获取当前文件描述符的标志
    flags = fcntl(fd, F_GETFL, 0);
    if (flags == -1) {
        perror("fcntl F_GETFL error after setting");
        return 1;
    }

    // 关闭文件
    if (close(fd) == -1) {
        perror("Failed to close file");
        return 1;
    }

    return 0;

}
```

根据测试结果可知，文件描述符的状态已由O_RDONLY修改为O_WRONLY

```
[  6.744172 0:10 linux_syscall_api::syscall_fs::imp::ctl:365] fd: 4, cmd: 3
[  6.744510 0:10 linux_syscall_api::syscall_fs::imp::ctl:393] get file status : 0
[  6.809889 0:10 linux_syscall_api::syscall_fs::imp::ctl:365] fd: 4, cmd: 4
[  6.810294 0:10 linux_syscall_api::syscall_fs::imp::ctl:365] fd: 4, cmd: 3
[  6.810529 0:10 linux_syscall_api::syscall_fs::imp::ctl:393] get file status : 1
        9:
        9:     calling fini:  [0]
        9:
        9:
        9:     calling fini: /lib/libc.so.6 [0]
        9:
        9:
        9:     calling fini: /lib64/ld-linux-x86-64.so.2 [0]
        9:
before change, File status flags (hex): 0
File descriptor 4 is now in non-blocking mode.
after change, File status flags (hex): 1
/ #
```