任务

`ioctl` 支持修改 `FIONBIO`：

- 如 `ioctl(4, FIONBIO, [1])` 表示修改 4 这个 fd 为非阻塞读写，`ioctl(5, FIONBIO, [0])` 表示修改 5 这个 fd 为阻塞读写。它等价于 open 时的 `O_NONBLOCK` 或者 fcntl 的 `F_SETFL`，可以参考这俩去实现。“设置为非阻塞读写”意味着当用户用 read 去读这个文件（实际上是个设备 `/sys/devices/system/cpu/online`）却没读到的时候，应该立即返回 EAGAIN，而不是卡在这里去切换其他进程。

概念

ioctl 是设备驱动程序中设备控制接口函数，一个字符设备驱动通常会实现设备打开、关闭、读、写等功能，在一些需要细分的情境下，如果需要扩展新的功能，通常以增设 ioctl() 命令的方式实现。

`ioctl` 函数的原型如下：

```
cCopy Codeint ioctl(int fd, unsigned long request, ...);
```

- `fd` 是打开设备文件的文件描述符。
- `request` 是要执行的命令或查询的请求代码，表示特定的操作。
- 可选的参数（使用变长参数列表）用于传递额外的参数给设备驱动程序。

通过使用不同的 `request` 值，可以执行各种设备控制操作，例如：

- 配置设备的属性和模式。
- 获取设备的状态信息。
- 控制设备的行为，如打开、关闭、重启等。
- 与设备进行数据交互，如读取或写入设备的数据。

需要注意的是，`ioctl` 是一个相对底层的系统调用，用于与设备驱动程序进行交互。在应用程序开发中，通常会使用更高级的接口和库来处理设备控制，而不是直接使用 `ioctl`。这些接口和库提供了更易用和跨平台的 API，使设备控制更加方便和可靠。

代码


api > arceos_posix_api > src > imp > fd_ops.rs > sys_fcntl
/// Manipulate file descriptor.
///
/// TODO: `SET/GET` command is ignored, hard-code stdin/stdout
pub fn sys_fcntl(fd: c_int, cmd: c_int, arg: usize) -> c_int {
debug!("sys_fcntl <= fd: {} cmd: {} arg: {}", fd, cmd, arg);
syscall_body!(sys_fcntl, {
match cmd as u32 {
ctypes::F_DUPFD => dup_fd(fd),
ctypes::F_DUPFD_CLOEXEC => {
// TODO: Change fd flags
dup_fd(fd)
}
ctypes::F_SETFL => {
if fd == 0 || fd == 1 || fd == 2 {
return Ok(0);
}
get_file_like(fd)?.set_nonblocking(arg & (ctypes::O_NONBLOCK as usize) > 0)?;
Ok(0)
}
_ => {
warn!("unsupported fcntl parameters: cmd {}", cmd);
Ok(0)
}
}
})
}

```rust
pub fn syscall_fcntl64(args: [usize; 6]) -> SyscallResult {
    let fd: usize = args[0];
    let cmd: usize = args[1];
    let arg: usize = args[2];
    let process: Arc<Process> = current_process();
    let mut fd_table: MutexGuard<Vec<Option<Arc<…>>>> = 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: Arc<dyn FileIO> = fd_table[fd].clone().unwrap();
    info!("fd: {}, cmd: {}", fd, cmd);
    match Fcntl64Cmd::try_from(cmd) {
        Ok(Fcntl64Cmd::F_DUPFD) => {
            let new_fd: usize = if let Ok(fd: usize) = 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(_is_set: (arg & 1) != 0) {
                Ok(0)
            } else {
                Err(SyscallError::EINVAL)
            }
        }
        Ok(Fcntl64Cmd::F_GETFL) => Ok(file.get_status().bits() as isize),
        Ok(Fcntl64Cmd::F_SETFL) => {
            if let Some(flags: OpenFlags) = OpenFlags::from_bits(arg as u32) {
                if file.set_status(flags) {
                    return Ok(0);
                }
            }
            Err(SyscallError::EINVAL)
        }
        Ok(Fcntl64Cmd::F_DUPFD_CLOEXEC) => {
            let new_fd: usize = if let Ok(fd: usize) = 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(_is_set: (arg & 1) != 0) {
                fd_table[new_fd] = fd_table[fd].clone();
```

第一版代码

```
match request {
        FIONBIO => {
            error!("in the FIONBIO option.");
            if fd == 0 || fd == 1 || fd == 2 {
                return Ok(0);
            }
            if argp == 0 {
                return Ok(0);
            }
            //设置非阻塞
            if let Some(flags) = OpenFlags::from_bits(argp as u32) {
                if file.set_status(flags) {
                    return Ok(0);
                }
            }
            Err(SyscallError::EAGAIN)
        }
        _ => Ok(0),
    }
```

问题：设置ioctl(5, FIONBIO, [0]) 和 ioctl(5, FIONBIO, [1]) 的效果一样

原因：argp并不是文件描述符的值，而是文件描述符的地址

```
        7:     file=libc.so.6 [0];  needed by ./ioctl [0]
        7:     file=libc.so.6 [0];  generating link map
        7:       dynamic: 0x0000000000021fbc0  base: 0x0000000000006000   size: 0x0000000000228e50
        7:         entry: 0x000000000002ff50  phdr: 0x0000000000006040  phnum:                 14
        7:
        7:
        7:     calling init: /lib64/ld-linux-x86-64.so.2
        7:
        7:
        7:     calling init: /lib/libc.so.6
        7:
        7:
        7:     initialize program: ./ioctl
        7:
        7:
        7:     transferring control: ./ioctl
        7:
[ 35.662185 0:8 axstarry::syscall_fs::imp::ctl:438] fd: 4, request: 21537, argp: 1073740848
[ 35.662351 0:8 axstarry::syscall_fs::imp::ctl:454] in the FIONBIO option.
ioctl: Resource temporarily unavailable
        7:
        7:     calling fini: ./ioctl [0]
        7:
```

解决方法：获取argp这个地址上的值，即为文件描述符的值

/ulib/axstarry/src/syscall_fs/imp/ctl.rs

```
pub fn syscall_ioctl(args: [usize; 6]) -> SyscallResult {
    let fd = args[0];
    let request = args[1];
    let argp = args[2];
    let process = current_process();
    let fd_table = process.fd_manager.fd_table.lock();
    info!("fd: {}, request: {}, argp: {}", fd, request, argp);
    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);
    }
    if process.manual_alloc_for_lazy(argp.into()).is_err() {
        return Err(SyscallError::EFAULT); // 地址不合法
    }
    let file = fd_table[fd].clone().unwrap();
    let ptr_argp = argp as *const u32;
    let nonblock = unsafe { ptr::read(ptr_argp) };
    match request {
        FIONBIO => {
            error!("in the FIONBIO option.");
            error!("nonblock: {}", nonblock);
            if fd == 0 || fd == 1 || fd == 2 {
                return Ok(0);
            }
            if nonblock == 1 {
                if let Some(flags) = OpenFlags::from_bits(nonblock as u32) {
                    if file.set_status(flags) {
                        return Ok(0);
                    }
                }
                return Err(SyscallError::EAGAIN);
            }
            Ok(0)
        }
        _ => Ok(0),
    }
}
```

至此，程序可以正常运行，在设置FIONBIO的flag为1时，进行非阻塞读写操作，flag为0时，进行阻塞读写操作

但是存在一个问题，目前的非阻塞操作设置flag为1后，立即返回EAGAIN，而不是在发生阻塞情况时返回EAGAIN，因此需要继续修改代码，添加阻塞情况的判断

这种方式相当于把syscall_ioctl写死了，应该直接调用各自 file 的 set_status，然后把 non_block 设置上

修改后代码

linux_syscall_api/src/syscall_fs/imp/ctl.rs

```
/// 执行各种设备相关的控制功能
/// todo: 未实现
/// # Arguments
/// * `fd`: usize, 文件描述符
/// * `request`: usize, 控制命令
/// * `argp`: *mut usize, 参数
pub fn syscall_ioctl(args: [usize; 6]) -> SyscallResult {
    let fd = args[0];
    let request = args[1];
    let argp = args[2];
    let process = current_process();
    let fd_table = process.fd_manager.fd_table.lock();
    info!("fd: {}, request: {}, argp: {}", fd, request, argp);
    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);
    }
    if process.manual_alloc_for_lazy(argp.into()).is_err() {
        return Err(SyscallError::EFAULT); // 地址不合法
    }

    let file = fd_table[fd].clone().unwrap();
    let ptr_argp = argp as *const u32;
    let nonblock = unsafe { ptr::read(ptr_argp) };
    match file.ioctl(request, argp) {
        Ok(ret) => Ok(ret),
        Err(_) => Ok(0),
    }
}
```

crates/axfs/src/api/port.rs

```
/// To control the file descriptor
    fn ioctl(&self, request: usize, data: usize) -> AxResult<isize> {
        match request {
            TIOCGWINSZ => {
                let winsize = data as *mut ConsoleWinSize;
                unsafe {
                    *winsize = ConsoleWinSize::default();
                }
                Ok(0)
            }
            TCGETS | TIOCSPGRP => {
                // pretend to be tty
                Ok(0)
            }

            TIOCGPGRP => {
                unsafe {
```

```
                    *(data as *mut u32) = 0;
                }
                Ok(0)
            }
            FIONBIO => {
                let ptr_argp = data as *const u32;
                let nonblock = unsafe { ptr::read(ptr_argp) };
                if nonblock == 1 {
                    let old_status = self.get_status();
                    let _ = self.set_status(old_status | OpenFlags::NON_BLOCK);
                }
                return Ok(0);
            }
            FIOCLEX => Ok(0),
            _ => Err(AxError::Unsupported),
        }
    }

}
```

测试用例

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#define FIONBIO 0x5421

int main() {
    int fd = open("file.txt", O_RDWR);
    if (fd < 0) {
        perror("Failed to open /dev/tty");
        return 1;
    }

    int arg = 1;
    if (ioctl(fd, FIONBIO, &arg) < 0) {
        perror("ioctl FIONBIO failed");
        close(fd);
        return 1;
    }

    int flags = fcntl(fd, F_GETFL, 0);
    if (flags < 0) {
        perror("fcntl F_GETFL failed");
        close(fd);
        return 1;
    }

    if (flags & O_NONBLOCK) {
        printf("File descriptor is non-blocking\n");
```

```
    } else {
        printf("File descriptor is blocking\n");
    }

    close(fd);
    return 0;
}
```

```
/ # ./ioctl
        9:     file=./ioctl [0];  generating link map
        9:       dynamic: 0x0000000000004d98  base: 0x0000000000001000   size: 0x0000000000004018
        9:         entry: 0x0000000000002120  phdr: 0x0000000000001040  phnum:                 13
        9:
        9:
        9:     file=libc.so.6 [0];  needed by ./ioctl [0]
        9:     file=libc.so.6 [0];  generating link map
        9:       dynamic: 0x00000000000221bc0  base: 0x0000000000008000   size: 0x0000000000228e50
        9:         entry: 0x0000000000031f50  phdr: 0x0000000000008040  phnum:                 14
        9:
        9:
        9:     calling init: /lib64/ld-linux-x86-64.so.2
        9:
        9:
        9:     calling init: /lib/libc.so.6
        9:
        9:
        9:     initialize program: ./ioctl
        9:
        9:
        9:     transferring control: ./ioctl
        9:
        9:
        9:     calling fini: ./ioctl [0]
        9:
File descriptor is non-blocking
```