Exercise 5.1

请根据kern/syscall_all.c 中的说明，完成sys_write_dev 函数以及sys_read_dev 函数的实现。

> 编写这两个系统调用时需要注意物理地址与内核虚拟地址之间的转换。
> 同时还要检查物理地址的有效性，在实验中允许访问的地址范围为:
> console:[0x10000000, 0x10000020), disk:[0x13000000, 0x13004200), rtc:[0x15000000,0x15000200)，当出现越界时，应返回指定的错误码.

```
/* Overview:
 *  This function is used to write data at 'va' with length 'len' to a device physical address
 *  'pa'. Remember to check the validity of 'va' and 'pa' (see Hint below);
 *
 *  'va' is the starting address of source data, 'len' is the
 *  length of data (in bytes), 'pa' is the physical address of
 *  the device (maybe with a offset).
 *
 * Post-Condition:
 *  Data within [va, va+len) is copied to the physical address 'pa'.
 *  Return 0 on success.
 *  Return -E_INVAL on bad address.
 *
 * Hint: Use the unmapped and uncached segment in kernel address space (KSEG1) to perform MMIO.
 * Hint: You can use 'is_illegal_va_range' to validate 'va'.
 * Hint: You MUST use 'memcpy' to copy data after checking the validity.
 *
 *  All valid device and their physical address ranges:
 *  * ---------------------------------*
 *  |   device   | start addr | length |
 *  * -----------+------------+--------*
 *  |  console   | 0x10000000 | 0x20   | (dev_cons.h)
 *  |  IDE disk  | 0x13000000 | 0x4200 | (dev_disk.h)
 *  |    rtc     | 0x15000000 | 0x200  | (dev_rtc.h)
 *  * ---------------------------------*
 */
int sys_write_dev(u_int va, u_int pa, u_int len) {
    /* Exercise 5.1: Your code here. (1/2) */
    if(is_illegal_va_range(va, len)) {
        return -E_INVAL;
    }


    if ((0x10000000 <= pa && pa + len <= 0x10000020) ||
        (0x13000000 <= pa && pa + len <= 0x13004200) ||
        (0x15000000 <= pa && pa + len <= 0x15000200)) {
        memcpy((void *)(pa + KSEG1), (void *)va, len);   //va的data复制到pa；注意pa + KSEG1转内核地址
        return 0;
```

```
    }

    return -E_INVAL;
}
```

```
/* Overview:
 *  This function is used to read data from a device physical address.
 *  Remember to check the validity of addresses (same as in 'sys_write_dev').
 *
 * Post-Condition:
 *  Data at 'pa' is copied from device to [va, va+len).
 *  Return 0 on success.
 *  Return -E_INVAL on bad address.
 *
 * Hint: You MUST use 'memcpy' to copy data after checking the validity.
 */
int sys_read_dev(u_int va, u_int pa, u_int len) {
    /* Exercise 5.1: Your code here. (2/2) */
    if(is_illegal_va_range(va, len)) {
        return -E_INVAL;
    }

    if ((0x10000000 <= pa && pa + len <= 0x10000020) ||
        (0x13000000 <= pa && pa + len <= 0x13004200) ||
        (0x15000000 <= pa && pa + len <= 0x15000200)) {
        memcpy((void *)va, (void *)(pa + KSEG1), len);    //复制到va里
        return 0;
    }

    return -E_INVAL;

}
```

Exercise 5.2

在user/lib/syscall_lib.c 中完成用户态的相应系统调用的接口。

```
int syscall_write_dev(void *va, u_int dev, u_int len) {
    /* Exercise 5.2: Your code here. (1/2) */
    return msyscall(SYS_write_dev, va, dev, len);
}

int syscall_read_dev(void *va, u_int dev, u_int len) {
    /* Exercise 5.2: Your code here. (2/2) */
    return msyscall(SYS_read_dev, va, dev, len);
}
```

Exercise 5.3

Exercise 5.3 参考以上展示的内核态驱动，使用**系统调用**完成fs/ide.c 中的ide_write函数，以及ide_read 函数，**实现对磁盘的读写操作。**

fs是用户态！ 要使用系统调用syscall

```
// Overview:
//  read data from IDE disk. First issue a read request through
//  disk register and then copy data from disk buffer
//  (512 bytes, a sector) to destination array.
//
// Parameters:
//  diskno: disk number.
//  secno: start sector number.
//  dst: destination for data read from IDE disk.
//  nsecs: the number of sectors to read.
//
// Post-Condition:
//  Panic if any error occurs. (you may want to use 'panic_on')
//
// Hint: Use syscalls to access device registers and buffers.
// Hint: Use the physical address and offsets defined in 'include/drivers/dev_disk.h':
//  'DEV_DISK_ADDRESS', 'DEV_DISK_ID', 'DEV_DISK_OFFSET', 'DEV_DISK_OPERATION_READ',
//  'DEV_DISK_START_OPERATION', 'DEV_DISK_STATUS', 'DEV_DISK_BUFFER'
void ide_read(u_int diskno, u_int secno, void *dst, u_int nsecs) {
    u_int begin = secno * BY2SECT;
    u_int end = begin + nsecs * BY2SECT;

    for (u_int off = 0; begin + off < end; off += BY2SECT) {
        uint32_t temp = diskno;
        /* Exercise 5.3: Your code here. (1/2) */
        panic_on(syscall_write_dev(&temp, DEV_DISK_ADDRESS | DEV_DISK_ID, 4)); //设置下一次读写的磁盘编号(写入diskno)
        uint32_t temp_off = begin + off;   //写入off是begin + off
        panic_on(syscall_write_dev(&temp_off, DEV_DISK_ADDRESS | DEV_DISK_OFFSET, 4));  //设置下一次磁盘镜像偏移字节数(写入off)
        u_int opt = DEV_DISK_OPERATION_READ;  // opt=0
        panic_on(syscall_write_dev(&opt, DEV_DISK_ADDRESS | DEV_DISK_START_OPERATION, 4)); //写入0开始读磁盘

        syscall_read_dev(&temp, DEV_DISK_ADDRESS | DEV_DISK_STATUS, 4);  //读上一次状态返回值(0是失败)
        if(temp == 0) {
            panic_on("fail to read.");
        }
        //读地址是dst+off
        panic_on(syscall_read_dev(dst+off, DEV_DISK_ADDRESS | DEV_DISK_BUFFER, BY2SECT)); //将对应off扇区的512bytes数据从设备缓冲区读入目标位置
    }
}
```

```c
// Overview:
//  write data to IDE disk.
//
// Parameters:
//  diskno: disk number.          磁盘号
//  secno: start sector number.   起始扇区号
//  src: the source data to write into IDE disk.   要写入IDE磁盘的源数据
//  nsecs: the number of sectors to write.
//
// Post-Condition:
//  Panic if any error occurs.
//
// Hint: Use syscalls to access device registers and buffers.   使用系统调用（因为fs是用户态下的进程）
// Hint: Use the physical address and offsets defined in 'include/drivers/dev_disk.h':
//  'DEV_DISK_ADDRESS', 'DEV_DISK_ID', 'DEV_DISK_OFFSET', 'DEV_DISK_BUFFER',
//  'DEV_DISK_OPERATION_WRITE', 'DEV_DISK_START_OPERATION', 'DEV_DISK_STATUS'
void ide_write(u_int diskno, u_int secno, void *src, u_int nsecs) {
    u_int begin = secno * BY2SECT;
    u_int end = begin + nsecs * BY2SECT;

    for (u_int off = 0; begin + off < end; off += BY2SECT) {
        uint32_t temp = diskno;
        /* Exercise 5.3: Your code here. (2/2) */
        uint32_t temp_off = begin + off;
        panic_on(syscall_write_dev(&temp, DEV_DISK_ADDRESS | DEV_DISK_ID, 4));
        panic_on(syscall_write_dev(&temp_off, DEV_DISK_ADDRESS | DEV_DISK_OFFSET, 4));
        panic_on(syscall_write_dev(src+off, DEV_DISK_ADDRESS | DEV_DISK_BUFFER, BY2SECT)); //先将对应扇区512bytes写入数据缓冲区
        u_int opt = DEV_DISK_OPERATION_WRITE;
        panic_on(syscall_write_dev(&opt, DEV_DISK_ADDRESS | DEV_DISK_START_OPERATION, 4)); //再写入1启动写磁盘操作
        syscall_read_dev(&temp, DEV_DISK_ADDRESS | DEV_DISK_STATUS, 4);
        if(temp == 0) {
            panic_on("fail to write.");
        }

    }
}
```

////// ↓ ↓ ↓ //////

Exercise 5.4

文件系统需要负责维护磁盘块的申请和释放，在回收一个磁盘块时，需要更改位图中的标志位。如果要将一个磁盘块设置为free，只需要将位图中对应位的值设置为1 即可。请完成fs/fs.c 中的free_block 函数，实现这一功能。同时思考为什么参数blockno 的值不能为0?

注：在常见的文件系统设计中，磁盘的编号通常从 1 开始，而不是从 0 开始。

```
// Overview:
// Mark a block as free in the bitmap.
void free_block(u_int blockno) {
    // You can refer to the function 'block_is_free' above.
    // Step 1: If 'blockno' is invalid (0 or >= the number of blocks in 'super'), return.
     // Step 2: Set the flag bit of 'blockno' in 'bitmap'.
      // Hint: Use bit operations to update the bitmap, such as b[n / W] |= 1 << (n % W).

}
```

```
// Overview:
//  Mark a block as free in the bitmap.
void free_block(u_int blockno) {
    // You can refer to the function 'block_is_free' above.
    // Step 1: If 'blockno' is invalid (0 or >= the number of blocks in 'super'), return.
    /* Exercise 5.4: Your code here. (1/2) */
    if(blockno == 0 || blockno >= super->s_nblocks) {    //super是超级块的指针，描述文件系统基本信息
        return;                // blockno不能是0
    }
    // Step 2: Set the flag bit of 'blockno' in 'bitmap'.
    // Hint: Use bit operations to update the bitmap, such as b[n / W] |= 1 << (n % W).
    /* Exercise 5.4: Your code here. (2/2) */
    bitmap[blockno / 32] |= (1 << (blockno % 32));  //bitmap数组每个元素含32bits，或上blockno对应32个里面的顺序(1是空闲)
}
```

Exercise 5.5

参照文件系统的设计，完成tools/fsformat.c 中的create_file 函数，并阅读write_directory 函数（代码已在源文件中给出，不作为考查点），实现将一个文件或指定目录下的文件按照目录结构写入到target/fs.img 的功能。

```
// Overview:
//  Allocate an unused 'struct File' under the specified directory.
//
//  Note that when we delete a file, we do not re-arrange all
//  other 'File's, so we should reuse existing unused 'File's here.
//
// Post-Condition:
//  Return a pointer to an unused 'struct File'.
//  We assume that this function will never fail.
//
// Hint:
//  Use 'make_link_block' to allocate a new block for the directory if there are no existing unused
```

```
//  'File's.
struct File *create_file(struct File *dirf) {
    int nblk = dirf->f_size / BY2BLK;

    // Step 1: Iterate through all existing blocks in the directory.
    for (int i = 0; i < nblk; ++i) {
        int bno; // the block number
        // If the block number is in the range of direct pointers (NDIRECT), get the 'bno'
        // directly from 'f_direct'. Otherwise, access the indirect block on 'disk' and get
        // the 'bno' at the index.
        /* Exercise 5.5: Your code here. (1/3) */
        if(i < NDIRECT) {
            bno = dirf->f_direct[i];  //获取块序号
        } else {
            bno = ((uint32_t* )(disk[dirf->f_indirect].data))[i];
            //dirf->f_indirect 是一个表示间接块在磁盘上位置的索引。disk[dirf->f_indirect].data 获取了间接块的数据。由于间接块中存储的是一系列块号（每个块号占用 uint32_t 的大小），因此我们需要将间接块的数据解释为 uint32_t 类型的指针，这样才能按索引 i 获取对应的块号。(把一整块数据转成一整组数组，然后数组名变指针)
        }
/***
struct Block {
    uint8_t data[BY2BLK];
    uint32_t type;
} disk[NBLOCK]
***/
        // Get the directory block using the block number.
        struct File *blk = (struct File *)(disk[bno].data);   //获取目录块。

        // Iterate through all 'File's in the directory block.
        for (struct File *f = blk; f < blk + FILE2BLK; ++f) {
            // If the first byte of the file name is null, the 'File' is unused.
            // Return a pointer to the unused 'File'.
            /* Exercise 5.5: Your code here. (2/3) */
            if(f->f_name[0] == '\0') {
                return f;
            }
        }

    }

    // Step 2: If no unused file is found, allocate a new block using 'make_link_block' function
    // and return a pointer to the new block on 'disk'.
    /* Exercise 5.5: Your code here. (3/3) */
    return (struct File* )(disk[make_link_block(dirf, nblk)].data);

    return NULL;
}
```

```c
// Overview:
//  Write directory to disk under specified dir.
//  Notice that we may use POSIX library functions to operate on
//  directory to get file infomation.
//
// Post-Condition:
//  We ASSUME that this funcion will never fail
void write_directory(struct File *dirf, char *path) {
    DIR *dir = opendir(path);
    if (dir == NULL) {
        perror("opendir");
        return;
    }
    struct File *pdir = create_file(dirf);
    strncpy(pdir->f_name, basename(path), MAXNAMELEN - 1);
    if (pdir->f_name[MAXNAMELEN - 1] != 0) {
        fprintf(stderr, "file name is too long: %s\n", path);
        // File already created, no way back from here.
        exit(1);
    }
    pdir->f_type = FTYPE_DIR;
    for (struct dirent *e; (e = readdir(dir)) != NULL;) {
        if (strcmp(e->d_name, ".") != 0 && strcmp(e->d_name, "..") != 0) {
            char *buf = malloc(strlen(path) + strlen(e->d_name) + 2);
            sprintf(buf, "%s/%s", path, e->d_name);
            if (e->d_type == DT_DIR) {
                write_directory(pdir, buf);
            } else {
                write_file(pdir, buf);
            }
            free(buf);
        }
    }
    closedir(dir);
}
```

Exercise 5.6

fs/fs.c 中的diskaddr 函数用来**计算指定磁盘块对应的虚存地址**。完成diskaddr 函数，根据一个块的序号(block number)，计算这一磁盘块对应的虚存的起始地址。 (提示：fs/serv.h 中的宏DISKMAP 和DISKMAX定义了磁盘映射虚存的地址空间)

```c
// Overview:
//  Return the virtual address of this disk block in cache.
// Hint: Use 'DISKMAP' and 'BY2BLK' to calculate the address.
void *diskaddr(u_int blockno) {
    /* Exercise 5.6: Your code here. */
    return DISKMAP + blockno * BY2BLK;
}

/***
#define DISKMAP 0x10000000
#define DISKMAX 0x40000000
将DISKMAP 到DISKMAP+DISKMAX 这一段虚存地址空间(0x10000000-0x4fffffff) 作为缓冲区。
***/
```

Exercise 5.7

实现fs/fs.c 中的map_block 函数，检查指定的磁盘块是否已经映射到内存，如果没有，分配一页内存来保存磁盘上的数据。相应地，完成unmap_block 函数，用于解除磁盘块和物理内存之间的映射关系，回收内存。（提示：注意磁盘虚拟内存地址空间和磁盘块之间的
对应关系）。

```c
// Overview:
//  Allocate a page to cache the disk block.
int map_block(u_int blockno) {
    // Step 1: If the block is already mapped in cache, return 0.
    // Hint: Use 'block_is_mapped'.
    /* Exercise 5.7: Your code here. (1/5) */
    if(block_is_mapped(blockno)) {
        return 0;
    }

    // Step 2: Alloc a page in permission 'PTE_D' via syscall.
    // Hint: Use 'diskaddr' for the virtual address.
    /* Exercise 5.7: Your code here. (2/5) */
    return syscall_mem_alloc(0, diskaddr(blockno), PTE_D);
          //使用系统调用分配内存，当前进程是env；将blockno转成对应虚拟地址
}

/***
void *block_is_mapped(u_int blockno) {
    void *va = diskaddr(blockno);
    if (va_is_mapped(va)) {
        return va;
    }
    return NULL;
}
***/
```

```c
// Overview:
//  Unmap a disk block in cache.
void unmap_block(u_int blockno) {
    // Step 1: Get the mapped address of the cache page of this block using 'block_is_mapped'.
    void *va;
    /* Exercise 5.7: Your code here. (3/5) */
    va = block_is_mapped(blockno);
    if(va == NULL) {
        return;
    }
    // Step 2: If this block is used (not free) and dirty in cache, write it back to the disk
    // first.
    // Hint: Use 'block_is_free', 'block_is_dirty' to check, and 'write_block' to sync.
    /* Exercise 5.7: Your code here. (4/5) */
    if(!block_is_free(blockno) && block_is_dirty(blockno)) {
        write_block(blockno);    //写磁盘块
    }

    // Step 3: Unmap the virtual address via syscall.
    /* Exercise 5.7: Your code here. (5/5) */
    syscall_mem_unmap(0, diskaddr(blockno));  //使用系统调用解除内存映射
    /* 也可写成 syscall_mem_unmap(0, va);  */

    user_assert(!block_is_mapped(blockno));
}
```

Exercise 5.8

补全fs/fs.c 中的dir_lookup 函数，查找某个目录下是否存在指定的文件。（使用file_get_block 函数）

```c
// Overview:
//  Find a file named 'name' in the directory 'dir'. If found, set *file to it.
//
// Post-Condition:
//  Return 0 on success, and set the pointer to the target file in `*file`.
//  Return the underlying error if an error occurs.
int dir_lookup(struct File *dir, char *name, struct File **file) {
    int r;
    // Step 1: Calculate the number of blocks in 'dir' via its size.
    u_int nblock;
    /* Exercise 5.8: Your code here. (1/3) */
    nblock = dir->f_size / BY2BLK;       //块数 = 文件总大小 ÷ 每块大小

    // Step 2: Iterate through all blocks in the directory.
    for (int i = 0; i < nblock; i++) {
        // Read the i'th block of 'dir' and get its address in 'blk' using 'file_get_block'.
        void *blk;
```

```
        /* Exercise 5.8: Your code here. (2/3) */
        try(file_get_block(dir, i, &blk));       //将某个指定文件指向的磁盘块读入内存
/***
int file_get_block(struct File *f, u_int filebno, void **blk) {
    int r;
    u_int diskbno;
    u_int isnew;
    // Step 1: find the disk block number is `f` using `file_map_block`.
    if ((r = file_map_block(f, filebno, &diskbno, 1)) < 0) {
        return r;
    }
    // Step 2: read the data in this disk to blk.
    if ((r = read_block(diskbno, blk, &isnew)) < 0) {
        return r;
    }
    return 0;
}
***/
        struct File *files = (struct File *)blk;

        // Find the target among all 'File's in this block.
        for (struct File *f = files; f < files + FILE2BLK; ++f) {
            // Compare the file name against 'name' using 'strcmp'.
            // If we find the target file, set '*file' to it and set up its
'f_dir'
            // field.
            /* Exercise 5.8: Your code here. (3/3) */
            if(strcmp(f->f_name, name) == 0) {
                *file = f;          //设置传入的file二级指针的内容
                f->f_dir = dir;     //设置文件所属的文件目录
                return 0;
            }

        }
    }
    return -E_NOT_FOUND;
}
```

Exercise 5.9

请完成user/lib/file.c 中的open 函数。（提示：若成功打开文件，则该函数返回文件描述符的编号）

```
// Overview:
//  Open a file (or directory).
//
// Returns:
//  the file descriptor on success,
//  the underlying error on failure.
int open(const char *path, int mode) {
    int r;

```

```
    // Step 1: Alloc a new 'Fd' using 'fd_alloc' in fd.c.
    // Hint: return the error code if failed.
    struct Fd *fd;
    /* Exercise 5.9: Your code here. (1/5) */
    try(fd_alloc(&fd));     //先分配一个fd（文件描述符）

    // Step 2: Prepare the 'fd' using 'fsipc_open' in fsipc.c.
    /* Exercise 5.9: Your code here. (2/5) */
    try(fsipc_open(path, mode, fd));  //将path对应文件的文件描述符共享到 fd 指针对应的地址处。

    // Step 3: Set 'va' to the address of the page where the 'fd''s data is cached, using
    // 'fd2data'. Set 'size' and 'fileid' correctly with the value in 'fd' as a 'Filefd'.
    char *va;
    struct Filefd *ffd;
    u_int size, fileid;
    /* Exercise 5.9: Your code here. (3/5) */
    va = fd2data(fd);             //获得地址
    ffd = (struct Filefd* )fd;     //将fa强制转换成filefd
    size = ffd->f_file.f_size;     //文件大小
    fileid = ffd->f_fileid;

    // Step 4: Alloc pages and map the file content using 'fsipc_map'.
    for (int i = 0; i < size; i += BY2PG) {
        /* Exercise 5.9: Your code here. (4/5) */
        try(fsipc_map(fileid, i, va+i));  // i即偏移量offset
/***
int fsipc_map(u_int fileid, u_int offset, void *dstva);
***/
    }

    // Step 5: Return the number of file descriptor using 'fd2num'.
    /* Exercise 5.9: Your code here. (5/5) */
    return fd2num(fd);    //返回文件描述符fd的编号
}
```

Exercise 5.10

参考user/lib/fd.c 中的write 函数，完成read 函数。

```
// Overview:
//  Read at most 'n' bytes from 'fd' at the current seek position into 'buf'.
//
// Post-Condition:
//  Update seek position.
//  Return the number of bytes read successfully.
//  Return < 0 on error.
int read(int fdnum, void *buf, u_int n) {
    int r;
```

```
    // Similar to the 'write' function below.
    // Step 1: Get 'fd' and 'dev' using 'fd_lookup' and 'dev_lookup'.
    struct Dev *dev;    // 磁盘设备
    struct Fd *fd;
    /* Exercise 5.10: Your code here. (1/4) */
    if((r = fd_lookup(fdnum, &fd)) < 0 || (r = dev_lookup(fd->fd_dev_id, &dev)) < 0) {
        return r;
    }
/***
int fd_lookup(int fdnum, struct Fd **fd);  Find the 'Fd' page for the given fd number.
int dev_lookup(int dev_id, struct Dev **dev);
***/
    // Step 2: Check the open mode in 'fd'.
    // Return -E_INVAL if the file is opened for writing only (O_WRONLY).
    /* Exercise 5.10: Your code here. (2/4) */
    if((fd->fd_omode & O_ACCMODE) == O_WRONLY) {
        return -E_INVAL;
    }

    // Step 3: Read from 'dev' into 'buf' at the seek position (offset in 'fd').
    /* Exercise 5.10: Your code here. (3/4) */
    r = dev->dev_read(fd, buf, n, fd->fd_offset);
/***Dev中读取(read)。    int (*dev_read)(struct Fd *, void *, u_int, u_int);
                        .dev_read = file_read,
                        static int file_read(struct Fd *fd, void *buf, u_int n, u_int offset);
***/

    // Step 4: Update the offset in 'fd' if the read is successful.
    /* Hint: DO NOT add a null terminator to the end of the buffer!
     *  A character buffer is not a C string. Only the memory within [buf, buf+n) is safe to
     *  use. */
    /* Exercise 5.10: Your code here. (4/4) */
    if(r > 0) {     // read成功则更新offset
        fd->fd_offset += r;
    }

    return r;
}
```

Exercise 5.11

完成fs/serv.c 中的serve_remove 函数。

```
// Overview:
//  Serve to remove a file specified by the path in `req`.
void serve_remove(u_int envid, struct Fsreq_remove *rq) {
```

```
    // Step 1: Remove the file specified in 'rq' using 'file_remove' and store
its return value.
    int r;
    /* Exercise 5.11: Your code here. (1/2) */
    r = file_remove(rq->req_path);   //file_remove定义在fs/fs.c

    // Step 2: Respond the return value to the requester 'envid' using
'ipc_send'.
    /* Exercise 5.11: Your code here. (2/2) */
    ipc_send(envid, r, 0, 0);     //使用ipc发送返回值
/***
void ipc_send(u_int whom, u_int value, const void *srcva, u_int perm);
***/
}
```

Exercise 5.12

完成user/lib/fsipc.c 中的fsipc_remove 函数。

```
// Overview:
//  Ask the file server to delete a file, given its path.
int fsipc_remove(const char *path) {
    // Step 1: Check the length of 'path' using 'strlen'.
    // If the length of path is 0 or larger than 'MAXPATHLEN', return -
E_BAD_PATH.
    /* Exercise 5.12: Your code here. (1/3) */
    if(strlen(path) == 0 || strlen(path) > MAXPATHLEN) {
        return -E_BAD_PATH;
    }

    // Step 2: Use 'fsipcbuf' as a 'struct Fsreq_remove'.
    struct Fsreq_remove *req = (struct Fsreq_remove *)fsipcbuf;

    // Step 3: Copy 'path' into the path in 'req' using 'strcpy'.
    /* Exercise 5.12: Your code here. (2/3) */
    strcpy(req->req_path, path);      //strcpy(dst, src);

    // Step 4: Send request to the server using 'fsipc'.
    /* Exercise 5.12: Your code here. (3/3) */
    return fsipc(FSREQ_REMOVE, req, 0, 0);
/***
static int fsipc(u_int type, void *fsreq, void *dstva, u_int *perm) {
    u_int whom;
    // Our file system server must be the 2nd env.
    ipc_send(envs[1].env_id, type, fsreq, PTE_D);
    return ipc_recv(&whom, dstva, perm);
}
***/
}
```

Exercise 5.13

完成user/lib/file.c 中的remove 函数。

```
// Overview:
//  Delete a file or directory.
int remove(const char *path) {   //定义删除文件的接口
    // Your code here.
    // Call fsipc_remove.

    /* Exercise 5.13: Your code here. */
    return fsipc_remove(path);  //调用ipc操作
}
```