浙江大学 2003 —2004 学年第一学期期终考试

《操作系统实验》课程试卷

考试时间：__30__ 分钟 开课学院：~~计算机学院~~ 专业：____________

姓名______________ 学号_____________ 任课教师_______________

题序	一	二	三	四	总分
评分					
评阅人					

PART II Operating System Lab Exam

1、 **Select the CORRECT and BEST answer for each of following questions and fill your answer in following blanks（30marks）**

1. ( ) 2. ( ) 3. ( ) 4. ( ) 5. ( )

6. ( ) 7. ( ) 8. ( ) 9. ( ) 10. ( )

11. ( ) 12. ( ) 13. ( ) 14. ( ) 15. ( )

1. I/O devices are treated as ________ in Linux.
A. common files B. directory files C. index files D. special files

2. The command "cat < test" _____________.
A. has the same result as " cat test"
B. creates a new file called test
C. redirects the output of cat
D. is not a completely valid command

3. The "!w" command will ________.
A. execute the last w command
B. execute the latest command that started with a w
C. both a and b
D. none of the above

4. Which one of the following is the correct command to store all of the contents of all files in the present working directory (or pwd) into a single new file called "all_in_one"? (assume that all_in_one does not exist anywhere in the current working directory and that the BASH shell is being used)
A. cat * > all_in_one
B. cp * all_in_one
C. cat * >> all_in_one
D. Both a and c

5. The command "wc -l *.c" will ________.
A. print the number of characters in all of the files ending in '.c'.
B. print the total number of lines in all the files ending in '.c' in the present working directory.
C. print the total # of words in all files ending in '.c' in the pwd.
D. None of the Above

6. If we want to install automatically a file system when power is up, which file can we modify?
A. /etc/mtab B. /etc/fastboot C./etc/fstab D./etc/inetd.conf

7. A program is reading a standard input from a keyboard. If you wish to terminate input and tell the system the end of input which combined keys can you type?
A. Ctrl+Z B. Ctrl+W C. Ctrl+D D. Ctrl+V

8. Following messages are displayed when *ps* command is executed. If you want to terminate process *bash*, which command can you use?

```
PID  TTY    TIME      CMD
336  pts/1  00:00:00  login
337  pts/1  00:00:00  bash
356  pts/1  00:00:00  ps
```

A. kill bash
B. kill pts/1
C. kill 337
D. kill !337

9. There are many kind of shell in Linux. Which is the least common used shell in following listed?
A. xsh
B. bash
C. ksh
D. csh

10. Which command is often used to build a Linux file system on a disk partition?
A. mknod
B. fdisk
C. format
D. mkfs

11. In order to make the file owner having *read* and *write* permissions while group and universal users can only *read* permission for the file, which of following octal number can we use together with *chmod* to change the file permission?
A. 566
B. 644
C. 655
D. 744

12. What is the default partition type in Linux?
A. vfat
B. ext2
C. swap
D. dos

13. To prevent to remove files by accident, which of options shall we use on *rm* command ?
A. -f
B. -R

C. -r
D. –i

14. Which of following commands can display the amount of disk space available on the file system A. du
B. df
C. mount
D. ln

15. What is the process number for following command?
```
$chmod 644 dir.txt&
[3] 164
```
A. 1
B. 3
C. 164
D. 644

二、Consider the following LINUX program, please show the possible output on display and in the file test.out. （**10 marks**）

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main()
{
   int fd;

   if( (fd=open("test.out", O_CREAT|O_RDWR, 0644)) == −1)
   {
      printf("Can not open the output file test.out\n");
      exit(−1);
   }
   printf("Begin to test\n");
   close(1);
   dup(fd);
   close(fd);
   printf("End of test\n");
   exit(0);
}
```

三、The following is the source code of Linux system call **sys_mount**()， Please briefly describe its functionality and draw the flow chart of **do_add_mount**() or **do_mount**()（**5 marks**）

```
struct file_system_type {
   const char *name;
   int fs_flags;
```

```
        struct super_block *(*read_super) (struct super_block *, void *, int);
        struct module *owner;
        struct file_system_type * next;
        struct list_head fs_supers;
};

static int do_add_mount(struct nameidata *nd, char *type, int flags,
                        int mnt_flags, char *name, void *data)
{
        struct vfsmount *mnt = do_kern_mount(type, flags, name, data);
        int err = PTR_ERR(mnt);

        if (IS_ERR(mnt))
                goto out;

        down(&mount_sem);
        /* Something was mounted here while we slept */
        while(d_mountpoint(nd->dentry) && follow_down(&nd->mnt, &nd->dentry))
                ;
        err = -EINVAL;
        if (!check_mnt(nd->mnt))
                goto unlock;

        /* Refuse the same filesystem on the same mount point */
        err = -EBUSY;
        if (nd->mnt->mnt_sb == mnt->mnt_sb && nd->mnt->mnt_root == nd->dentry)
                goto unlock;

        mnt->mnt_flags = mnt_flags;
        err = graft_tree(mnt, nd);
unlock:
        up(&mount_sem);
        mntput(mnt);
out:
        return err;
}

/*
 * Flags is a 32-bit value that allows up to 31 non-fs dependent flags to
 * be given to the mount() call (ie: read-only, no-dev, no-suid etc).
 *
 * data is a (void *) that can point to any structure up to
 * PAGE_SIZE-1 bytes, which can contain arbitrary fs-dependent
 * information (or be NULL).
 *
*/
long do_mount(char * dev_name, char * dir_name, char *type_page,
              unsigned long flags, void *data_page)
{
        struct nameidata nd;
        int retval = 0;
        int mnt_flags = 0;

        /* Discard magic */
        if ((flags & MS_MGC_MSK) == MS_MGC_VAL)
                flags &= ~MS_MGC_MSK;

        /* Basic sanity checks */

        if (!dir_name || !*dir_name || !memchr(dir_name, 0, PAGE_SIZE))
                return -EINVAL;
        if (dev_name && !memchr(dev_name, 0, PAGE_SIZE))
                return -EINVAL;

        /* Separate the per-mountpoint flags */
        if (flags & MS_NOSUID)
                mnt_flags |= MNT_NOSUID;
        if (flags & MS_NODEV)
                mnt_flags |= MNT_NODEV;
        if (flags & MS_NOEXEC)
                mnt_flags |= MNT_NOEXEC;
        flags &= ~(MS_NOSUID|MS_NOEXEC|MS_NODEV);

        /* ... and get the mountpoint */
        if (path_init(dir_name, LOOKUP_FOLLOW|LOOKUP_POSITIVE, &nd))
                retval = path_walk(dir_name, &nd);
        if (retval)
                return retval;

        if (flags & MS_REMOUNT)
                retval = do_remount(&nd, flags & ~MS_REMOUNT, mnt_flags,
                                  data_page);
        else if (flags & MS_BIND)
                retval = do_loopback(&nd, dev_name, flags & MS_REC);
        else if (flags & MS_MOVE)
                retval = do_move_mount(&nd, dev_name);
        else
                retval = do_add_mount(&nd, type_page, flags, mnt_flags,
                                        dev_name, data_page);
        path_release(&nd);
        return retval;
}

asmlinkage long sys_mount(char * dev_name, char * dir_name, char * type,
                          unsigned long flags, void * data)
{
        int retval;
        unsigned long data_page;
        unsigned long type_page;
        unsigned long dev_page;
        char *dir_page;

        retval = copy_mount_options (type, &type_page);
        if (retval < 0)
                return retval;

        dir_page = getname(dir_name);
        retval = PTR_ERR(dir_page);
        if (IS_ERR(dir_page))
                goto out1;

        retval = copy_mount_options (dev_name, &dev_page);
        if (retval < 0)
                goto out2;

        retval = copy_mount_options (data, &data_page);
        if (retval < 0)
                goto out3;

        lock_kernel();
        retval = do_mount((char*)dev_page, dir_page, (char*)type_page,
                          flags, (void*)data_page);
        unlock_kernel();
        free_page(data_page);

out3:
        free_page(dev_page);
out2:
        putname(dir_page);
out1:
        free_page(type_page);
        return retval;
```

}

四、The following is part of the source code for Linux virtual memory allocation function **vmalloc**(). Please briefly describe its functionality and draw the flow chart depending on given source code（**5marks**）

```
/*
* vmalloc() function
 */
struct vm_struct {
        unsigned long flags;      /* virtual memory block state flag */
        void * addr;              /* pointer to the virtual memory block */
        unsigned long size;       /* block size */
        struct vm_struct * next;  /* pointer to the next block */
};
static struct vm_struct * vmlist = NULL;


rwlock_t vmlist_lock = RW_LOCK_UNLOCKED;
struct vm_struct * vmlist;

inline int vmalloc_area_pages (unsigned long address, unsigned long size,
                        int gfp_mask, pgprot_t prot)
{
    pgd_t * dir;
    unsigned long end = address + size;
    int ret;

    dir = pgd_offset_k(address); /* to find an entry in a kernel page-table-directory */
    spin_lock(&init_mm.page_table_lock);
    do {
        pmd_t *pmd;

        pmd = pmd_alloc(&init_mm, dir, address);
        ret = -ENOMEM;
        if (!pmd)
            break;

        ret = -ENOMEM;
        if (alloc_area_pmd(pmd, address, end - address, gfp_mask, prot))
            break;

        address = (address + PGDIR_SIZE) & PGDIR_MASK;
        dir++;

        ret = 0;
    } while (address && (address < end));
```

```
        spin_unlock(&init_mm.page_table_lock);
        flush_cache_all();
        return ret;
}

struct vm_struct * get_vm_area(unsigned long size, unsigned long flags)
{
        unsigned long addr;
        struct vm_struct **p, *tmp, *area;

        area = (struct vm_struct *) kmalloc(sizeof(*area), GFP_KERNEL);
        if (!area)
                return NULL;
        size += PAGE_SIZE;
        addr = VMALLOC_START;
        write_lock(&vmlist_lock);/* write spinlock */
        for (p = &vmlist; (tmp = *p) ; p = &tmp->next) {
                if ((size + addr) < addr)
                        goto out;
                if (size + addr <= (unsigned long) tmp->addr)
                        break;
                addr = tmp->size + (unsigned long) tmp->addr;
                if (addr > VMALLOC_END-size)
                        goto out;
        }
        area->flags = flags;
        area->addr = (void *)addr;
        area->size = size;
        area->next = *p;
        *p = area;
        write_unlock(&vmlist_lock);     return area;

out:
        write_unlock(&vmlist_lock);
        kfree(area);
        return NULL;
}

void * __vmalloc (unsigned long size, int gfp_mask, pgprot_t prot)
{
        void * addr;
        struct vm_struct *area;

        size = PAGE_ALIGN(size);
        if (!size || (size >> PAGE_SHIFT) > num_physpages) {
                BUG();
                return NULL;
        }
        area = get_vm_area(size, VM_ALLOC);
        if (!area)
                return NULL;
        addr = area->addr;
        if (vmalloc_area_pages(VMALLOC_VMADDR(addr), size, gfp_mask, prot)) {
                vfree(addr);
                return NULL;
        }
        return addr;
}

/*
 *      Allocate any pages
 */

static inline void * vmalloc (unsigned long size)
{
        return __vmalloc(size, GFP_KERNEL | __GFP_HIGHMEM, PAGE_KERNEL);
}
```