

# 实验七

## *Hollow Man*

### 实验名称：

存储管理

### 实验目的：

1. 观察系统存储器使用情况
2. 观察进程使用存储器的情况
3. 掌握通过内存映像文件提高性能的方法
4. 掌握动态内存分配技术

### 实验时间

3 学时

### 预备知识：

1. 存储相关的命令

- 1.1 free 显示系统使用和未被使用的内存数量（可以实时执行）

输出包含的标题有 3 行信息：

**Mem。**此行包含了有关物理内存的信息。包括以下详细内容：

**total。**该项显示可用的物理内存总量，单位为 **KB**。该数字小于安装的物理内存的容量，是因为内核本身也要使用一小部分的内存。

**used。**该项显示了用于应用程序超速缓存数据的内存容量。

**free。**该项显示了此时未使用且有效的内存容量。

**Shared/buffers/cached。**这些列显示了有关内存如何使用的更为详细的信息。

**-/+ buffers/cache。**Linux 系统中的部分内存用来为应用程序或设备高速缓存数据。这部分内存存在需要用于其他目的时可以释放。

**free** 列显示了调整的缓冲区行，显示释放缓冲区或高速缓存时可以使用的内存容量。

**Swap。**该行显示有关交换内存利用率的信息。该信息包含全部、已使用和释放的可用内存容量。

- 1.2 vmstat 报告进程、内存、分页、IO 等多类信息（使用手册页）

- 1.3 size 列出目标文件段大小和总大小（使用手册页）

## 2. /proc 文件系统（使用手册页 man 5 proc）

### 2.1 /proc/meminfo 内存状态信息

### 2.2 /proc/stat 包含内存页、内存对换等信息。

### 2.3 /proc/\$pid/stat 某个进程的信息(包含内存使用信息)

### 2.4 /proc/\$pid/maps 某个进程的内存映射区信息，包括地址范围、权限、偏移量以及主次设备号和映射文件的索引节点。

### 2.5 /proc/\$pid/statm 某个进程的内存使用信息，包括内存总大小、驻留集大小、共享页面数、文本页面数、堆栈页面数和脏页面数。

## 3. 内存映像文件

内存映像文件是指把一个磁盘文件映像到内存中，二者存在逐字节的对应关系。这样做可以加速 I/O 操作，并可以共享数据。

### 3.1 mmap（建立内存映射）

表头文件 `#include <unistd.h>`

`#include <sys/mman.h>`

定义函数 `void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offsize);`

函数说明 `mmap()`用来将某个文件内容映射到内存中，对该内存区域的存取即是直接对该文件内容的读写。参数 `start` 指向欲对应的内存起始地址，通常设为 `NULL`，代表让系统自动选定地址，对应成功后该地址会返回。参数 `length` 代表将文件中多大的部分对应到内存。

参数 `prot` 代表映射区域的保护方式有下列组合

`PROT_EXEC` 映射区域可被执行

`PROT_READ` 映射区域可被读取

`PROT_WRITE` 映射区域可被写入

`PROT_NONE` 映射区域不能存取

参数 `flags` 会影响映射区域的各种特性

`MAP_FIXED` 如果参数 `start` 所指的地址无法成功建立映射时，则放弃映射，不对地址做修正。通常不鼓励用此旗标。

`MAP_SHARED` 对映射区域的写入数据会复制回文件内，而且允许其他映射该文件的进程共享。

`MAP_PRIVATE` 对映射区域的写入操作会产生一个映射文件的复制，即私人的“写入时复制”（copy on write）对此区域作的任何修改都不会写回原来的文件内容。

`MAP_ANONYMOUS` 建立匿名映射。此时会忽略参数 `fd`，不涉及文件，而且映射

区域无法和其他进程共享。

MAP\_DENYWRITE 只允许对映射区域的写入操作，其他对文件直接写入的操作将会被拒绝。

MAP\_LOCKED 将映射区域锁定住，这表示该区域不会被置换（swap）。

在调用 mmap()时必须指定 MAP\_SHARED 或 MAP\_PRIVATE。参数 fd 为 open()返回的文件描述词，代表欲映射到内存的文件。参数 offset 为文件映射的偏移量，通常设置为 0，代表从文件最前方开始对应，offset 必须是分页大小的整数倍。

返回值 若映射成功则返回映射区的内存起始地址，否则返回 MAP\_FAILED(-1)，错误原因存于 errno 中。

错误代码 EBADF 参数 fd 不是有效的文件描述词

EACCES 存取权限有误。如果是 MAP\_PRIVATE 情况下文件必须可读，使用 MAP\_SHARED 则要有 PROT\_WRITE 以及该文件要能写入。

EINVAL 参数 start、length 或 offset 有一个不合法。

EAGAIN 文件被锁住，或是有太多内存被锁住。

ENOMEM 内存不足。

### 3.2 munmap（解除内存映射）

表头文件 #include<unistd.h>

#include<sys/mman.h>

定义函数 int munmap(void \*start,size\_t length);

函数说明 munmap()用来取消参数 start 所指的映射内存起始地址，参数 length 则是欲取消的内存大小。当进程结束或利用 exec 相关函数来执行其他程序时，映射内存会自动解除，但关闭对应的文件描述词时不会解除映射。

返回值 如果解除映射成功则返回 0，否则返回 -1，错误原因存于 errno 中错误代码 EINVAL

参数 start 或 length 不合法。

## 4. 动态内存分配

### 4.1 malloc（配置内存空间）

表头文件 #include<stdlib.h>

定义函数 void \* malloc(size\_t size);

函数说明 malloc()用来配置内存空间，其大小由指定的 size 决定。

返回值 若配置成功则返回一指针，失败则返回 NULL。

### 4.2 free（释放原先配置的内存）

表头文件 `#include<stdlib.h>`

定义函数 `void free(void *ptr);`

函数说明 参数 `ptr` 为指向先前由 `malloc()`、`calloc()`或 `realloc()`所返回的内存指针。调用 `free()`后 `ptr` 所指的内存空间便会被收回。假若参数 `ptr` 所指的内存空间已被收回或是未知的内存地址，则调用 `free()`可能会有无法预期的情况发生。若参数 `ptr` 为 `NULL`，则 `free()`不会有任何作用。

#### 4.3 `calloc`（配置内存空间）

表头文件 `#include <stdlib.h>`

定义函数 `void *calloc(size_t nmemb, size_t size);`

函数说明 `calloc()`用来配置 `nmemb` 个相邻的内存单位，每一单位的大小为 `size`，并返回指向第一个元素的指针。这和使用下列的方式效果相同：`malloc(nmemb*size);`不过，在利用 `calloc()`配置内存时会将内存内容初始化为 0。

返回值 若配置成功则返回一指针，失败则返回 `NULL`。

#### 5. 其他

`getpagesize`（取得内存分页大小）

表头文件 `#include<unistd.h>`

定义函数 `size_t getpagesize(void);`

函数说明 返回一分页的大小，单位为字节（byte）。此为系统的分页大小，不一定会和硬件分页大小相同。

返回值 内存分页大小。附加说明在 Intel x86 上其返回值应为 4096bytes。

#### 6. `fstat`（由文件描述词取得文件状态）

表头文件 `#include<sys/stat.h>`

`#include<unistd.h>`

定义函数 `int fstat(int fildes, struct stat *buf);`

函数说明 `fstat()`用来将参数 `fildes` 所指的文件状态，复制到参数 `buf` 所指的结构中（`struct stat`）。`fstat()`与 `stat()`作用完全相同，不同处在于传入的参数为已打开的文件描述词。

返回值 执行成功则返回 0，失败返回-1，错误代码存于 `errno`。

`struct stat;`

```

struct stat {
    dev_t      st_dev;      /* device */
    ino_t      st_ino;      /* inode */
    mode_t     st_mode;     /* protection */
    nlink_t    st_nlink;    /* number of hard links */
    uid_t      st_uid;      /* user ID of owner */
    gid_t      st_gid;      /* group ID of owner */
    dev_t      st_rdev;     /* device type (if inode device) */
    off_t      st_size;     /* total size, in bytes */
    unsigned long st_blksize; /* blocksize for filesystem I/O */
    unsigned long st_blocks; /* number of blocks allocated */
    time_t     st_atime;    /* time of last access */
    time_t     st_mtime;    /* time of last modification */
    time_t     st_ctime;    /* time of last change */
};

```

列数种情况

S\_IFMT 0170000 文件类型的位掩码  
 S\_IFSOCK 0140000 socket  
 S\_IFLNK 0120000 符号连接  
 S\_IFREG 0100000 一般文件  
 S\_IFBLK 0060000 区块装置  
 S\_IFDIR 0040000 目录  
 S\_IFCHR 0020000 字符装置  
 S\_IFIFO 0010000 先进先出  
 S\_ISUID 04000 文件的 (set user-id on execution) 位  
 S\_ISGID 02000 文件的 (set group-id on execution) 位  
 S\_ISVTX 01000 文件的 sticky 位  
 S\_IRUSR (S\_IREAD) 00400 文件所有者具可读取权限  
 S\_IWUSR (S\_IWRITE) 00200 文件所有者具可写入权限  
 S\_IXUSR (S\_IEXEC) 00100 文件所有者具可执行权限  
 S\_IRGRP 00040 用户组具可读取权限  
 S\_IWGRP 00020 用户组具可写入权限  
 S\_IXGRP 00010 用户组具可执行权限  
 S\_IROTH 00004 其他用户具可读取权限  
 S\_IWOTH 00002 其他用户具可写入权限  
 S\_IXOTH 00001 其他用户具可执行权限

上述的文件类型在 POSIX 中定义了检查这些类型的宏定义

S\_ISLNK (st\_mode) 判断是否为符号连接

S\_ISREG (st\_mode) 是否为一般文件

S\_ISDIR (st\_mode) 是否为目录

S\_ISCHR (st\_mode) 是否为字符装置文件

S\_ISBLK (s3e) 是否为先进先出

S\_ISSOCK (st\_mode) 是否为 socket

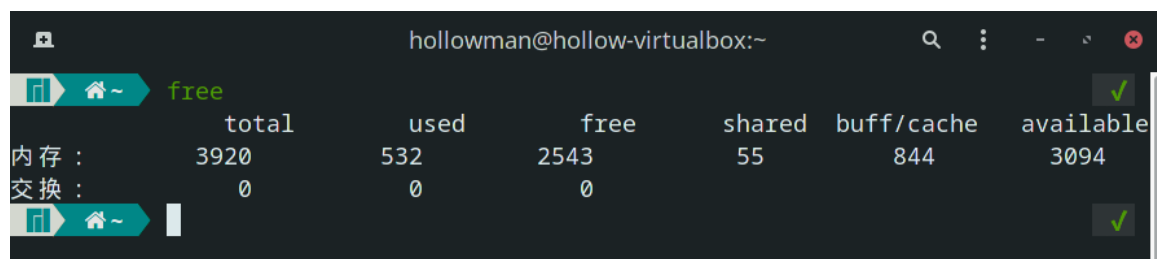
范例

```
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
main()
{
    struct stat buf;
    int fd;
    fd = open ( "/etc/passwd" ,O_RDONLY);
    fstat(fd,&buf);
    printf( "/etc/passwd file size +%d\n ",buf.st_size);
}
```

## 实验要求:

1. 分别使用命令和/proc 文件系统列出系统当前内存的使用情况。

使用 free 命令查看内存使用情况:



	total	used	free	shared	buff/cache	available
内存 :	3920	532	2543	55	844	3094
交换 :	0	0	0			

打开 proc/meminfo 文件查看内存使用情况:



```
hollowman@hollow-virtualbox:~  
free  
total        used        free        shared  buff/cache   available  
内存：      3920        539        2121         55        1259        3086  
交换：         0          0          0  
free  
total        used        free        shared  buff/cache   available  
内存：      3920        552        2055         55        1313        3073  
交换：         0          0          0  
free  
total        used        free        shared  buff/cache   available  
内存：      3920        679        1853         63        1386        2937  
交换：         0          0          0  
vmstat  
procs -----memory----- --swap--  -----io----- -system--  -----cpu-----  
r  b 交换 空闲 缓冲 缓存  si  so  bi  bo  in  cs us sy id wa st  
2  1      0 1586648 53060 1455656    0    0  361  19 110 154 1 2 91 6  
0  
free  
total        used        free        shared  buff/cache   available  
内存：      3920        560        1938         55        1420        3063  
交换：         0          0          0  
vmstat  
procs -----memory----- --swap--  -----io----- -system--  -----cpu-----  
r  b 交换 空闲 缓冲 缓存  si  so  bi  bo  in  cs us sy id wa st  
1  0      0 1986024 53380 1401448    0    0  343  20 112 160 1 2 91 6  
0
```

3. 用 `size` 工具观察三个不同的可执行文件的大小以及它们段的大小。  
Size 命令的输出不包括 `stack` 和 `heap` 的部分。只包括文本段 (`text`)，代码段 (`data`)，未初始化数据段 (`bss`) 三部分。

```
hollowman@hollow-virtualbox:~  
size /usr/bin/grep  
text  data  bss  dec  hex filename  
150430 4336 68424 223190 367d6 /usr/bin/grep  
size /usr/bin/cat  
text  data  bss  dec  hex filename  
29611 1576 408 31595 7b6b /usr/bin/cat  
size /usr/bin/ls  
text  data  bss  dec  hex filename  
127723 4744 4824 137291 2184b /usr/bin/ls
```

4. 启动一个耗时较长的后台进程，通过 `/proc` 文件系统查看该进程所有内存使用相关信息，并列出。



```
hollowman@hollow-virtualbox:~  
ps aux | grep firefox  
hollowm+ 2379 72.5 7.5 2760092 302664 ? S1 09:00 0:05 /usr/lib/firefox/firefox --new-window  
hollowm+ 2436 23.6 5.1 2640748 206416 ? S1 09:00 0:01 /usr/lib/firefox/firefox -contentproc -childID 1 -isForBrowser -prefslen 1 -prefMapSize 234694 -parentBuildID 20210506154205 -appdir /usr/lib/firefox/browser 2379 true tab  
hollowm+ 2495 17.2 3.2 2478620 131192 ? S1 09:00 0:00 /usr/lib/firefox/firefox -contentproc -childID 2 -isForBrowser -prefslen 4936 -prefMapSize 234694 -parentBuildID 20210506154205 -appdir /usr/lib/firefox/browser 2379 true tab  
hollowm+ 2538 17.5 3.6 2479964 147112 ? S1 09:00 0:00 /usr/lib/firefox/firefox -contentproc -childID 3 -isForBrowser -prefslen 5764 -prefMapSize 234694 -parentBuildID 20210506154205 -appdir /usr/lib/firefox/browser 2379 true tab  
hollowm+ 2578 17.0 2.2 2443356 91516 ? S1 09:00 0:00 /usr/lib/firefox/firefox -contentproc -childID 4 -isForBrowser -prefslen 5830 -prefMapSize 234694 -parentBuildID 20210506154205 -appdir /usr/lib/firefox/browser 2379 true tab  
hollowm+ 2605 0.0 0.0 9500 2336 pts/0 S+ 09:00 0:00 grep firefox  
sudo cat /proc/2578/maps  
[sudo] hollowman 的密码 :  
251aa3832000-251aa3842000 ---p 00000000 00:00 0  
251aa3842000-251aa385f000 r-xp 00000000 00:00 0  
251aa385f000-251aa3862000 r-xp 00000000 00:00 0  
251aa3862000-251aa3872000 ---p 00000000 00:00 0  
251aa3872000-251aa3875000 r-xp 00000000 00:00 0  
251aa3875000-251aa3882000 r-xp 00000000 00:00 0
```

5. 编写一个程序，打印系统的页面大小。

程序代码：

```
#include <stdio.h>  
#include <unistd.h>  
int main() {  
    printf("page-size:%d\n", getpagesize());  
}
```

```
hollowman@hollow-virtualbox:~/文档  
nano 5.c  
gcc 5.c  
./a.out  
page-size:4096
```

6. 阅读并编译运行以下程序，总结内存映象文件的使用方法。

范例 /\* 利用 mmap()来读取/etc/passwd 文件内容\*/

```
#include<sys/types.h>
```

```
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<sys/mman.h>
main()
{
    int fd;
    void *start;
    struct stat sb;
    fd=open( “/etc/passwd” ,O_RDONLY); /*打开/etc/passwd*/
    fstat(fd,&sb); /*取得文件大小*/
    /* 利用 man fstat 可以看到 struct stat 的定义*/
    start=mmap(NULL,sb.st_size,PROT_READ,MAP_PRIVATE,fd,0);
    if(start== MAP_FAILED) /*判断是否映射成功*/
        return;
    printf(“%s”,start);
    munmap(start,sb.st_size); /*解除映射*/
    close (fd);
}
```

```
hollowman@hollow-virtualbox:~/文档
nano 6.c
gcc 6.c
6.c:7:1: 警告：返回类型默认为 'int' [-Wimplicit-int]
7 | main()
  | ^~~~
6.c: 在函数 'main' 中:
6.c:17:5: 警告：在有返回值的函数中，'return' 不带返回值
17 |     return;
    |     ^~~~~~
6.c:7:1: 附注：在此声明
7 | main()
  | ^~~~
6.c:18:5: 警告：隐式声明函数 'printf' [-Wimplicit-function-declaration]
18 |     printf("%s", start);
    |     ^~~~~~
6.c:18:5: 警告：隐式声明与内建函数 'printf' 不兼容
6.c:7:1: 附注：include '<stdio.h>' or provide a declaration of 'printf'
6 | #include<sys/mman.h>
+++ |+#include <stdio.h>
7 | main()
./a.out
root:x:0:0::/root:/bin/bash
nobody:x:65534:65534:Nobody:/:/usr/bin/nologin
dbus:x:81:81:System Message Bus:/:/usr/bin/nologin
bin:x:1:1:/:/usr/bin/nologin
daemon:x:2:2:/:/usr/bin/nologin
mail:x:8:12:/:var/spool/mail:/usr/bin/nologin
ftp:x:14:11:/:srv/ftp:/usr/bin/nologin
http:x:33:33:/:srv/http:/usr/bin/nologin
systemd-journal-remote:x:982:982:systemd Journal Remote:/:/usr/bin/nologin
systemd-network:x:981:981:systemd Network Management:/:/usr/bin/nologin
systemd-resolve:x:980:980:systemd Resolver:/:/usr/bin/nologin
```

总结内存映象文件的使用方法：

内存映像其实就是在内存中创建一个和外存文件完全相同的映像，用户可以将整个文件映射到内存，也可以部分映射。通过内存映像实现对外存文件的操作。首先 Mmap 申请虚拟内存，再次调用 file 指针所指映射函数对其进行映射。判断是否映射成功。进行操作。最后解除映射。

7. 编写一个程序，利用内存映象文件，实现 less 工具的功能。

程序代码如下：

```
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <string.h>
#include <memory.h>
```

```

#include <stdlib.h>
#include <stdio.h>
int lastrow(char *s, int d);
int nextrow(char *s, int d);
int onepage(char *s, int d);
int main(int argc, char *argv[])
{
    if( argc == 2 ) // 判断参数是否只有一个
    {
        int fd, play = 0;
        char lab;
        char *start;
        struct stat sb;
        fd = open(argv[1], O_RDONLY); // 以只读方式打开文件
        fstat(fd, &sb); // 获取文件的大小
        start = mmap(NULL, sb.st_size, PROT_READ, MAP_PRIVATE, fd, 0);
        if (start == MAP_FAILED) // MAP_FAILED 表示映射失败
            return (1);
        play = onepage(start, play) + 1;
        lab = getchar();
        while (lab != 'q') // 输入的字符为 q，退出
        {
            if (play > sb.st_size) // 如果 onepage 返回的字节数大于文件的大小，输入任意字符退出
            {
                lab = getchar();
                break;
            }
            else if (lab == 'p') // 输入 p，继续读 10 行
                play += onepage(start, play) + 1;
            else if (lab == 'n') // 输入 n，显示下一行
                play += nextrow(start, play) + 1;
            else if (lab == 'l') // 输入 l，显示上一行
                play = lastrow(start, play) + 1;
            lab = getchar();
        }
        munmap(start, sb.st_size); // 解除映射
        close(fd); // 关闭文件 fd
        return 0;
    }
    else if( argc > 2 )
    {
        printf("Too many arguments supplied.\n");
        return 1;
    }
    else
    {
        printf("One argument expected.\n");
        return 1;
    }
}
int onepage(char *s, int d)

```

```

{
    int i, count = 0; // count 在这里表示文件中行的数量
    char *buffer = malloc(2048); // 配置内存空间，由 buffer 指向该空间

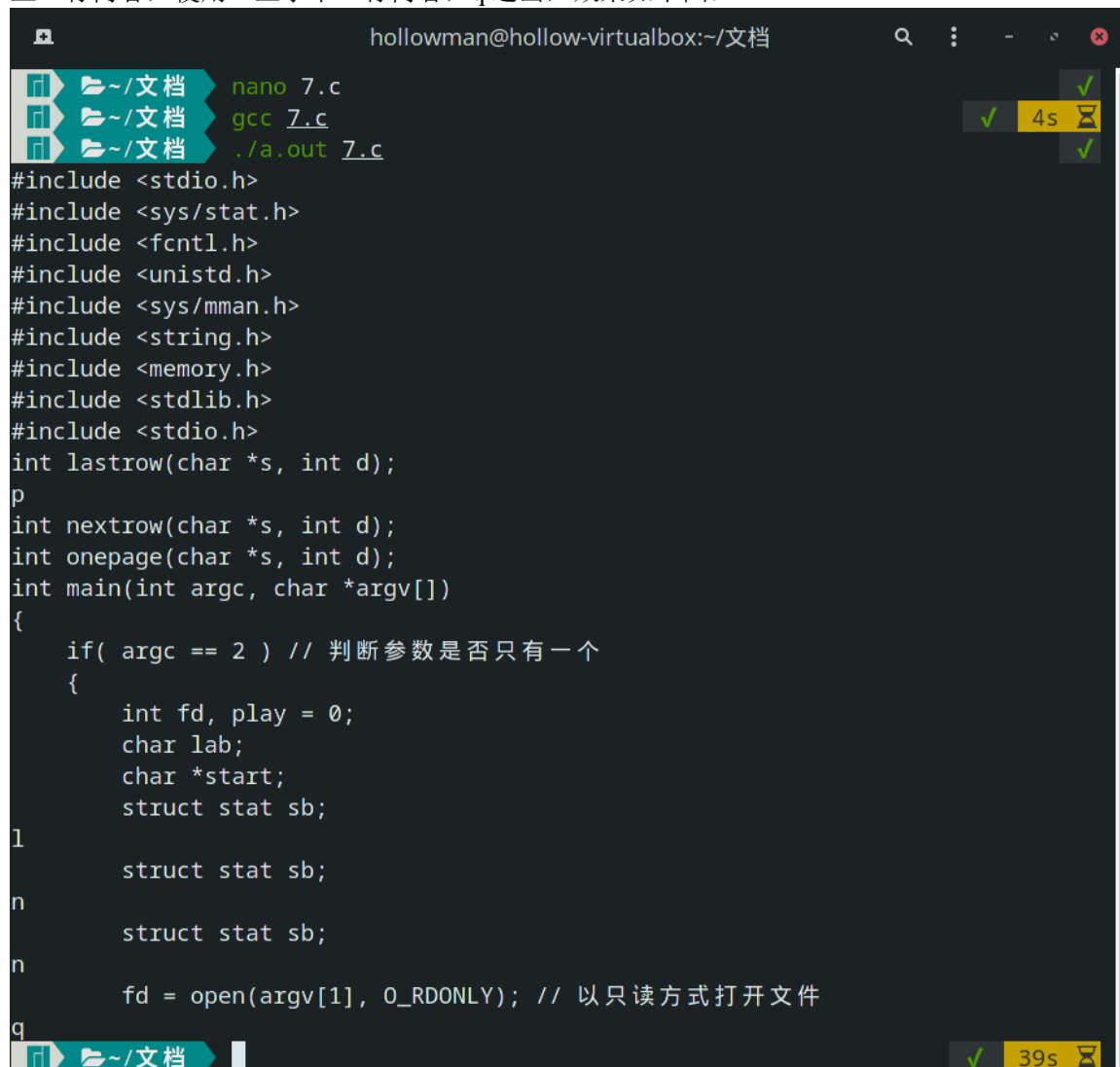
    s += d; // 每 10 行作为一页输出
    for (i = 0; i < 2048; i++)
    {
        if (s[i] == '\n')
            count++;
        if (count == 10)
            break;
    }
    memcpy(buffer, s, i); // 从 s 处开始的地方拷贝 i 个字节到 buffer
    buffer[i] = '\0'; // 添加结束标识
    printf("%s\n", buffer);
    return i;
}
int nextrow(char *s, int d) // 下一行
{
    int i;
    char *buffer = malloc(100);
    s += d;
    for (i = 0; i < 100; i++)
        if (s[i] == '\n')
            break;
    memcpy(buffer, s, i);
    buffer[i] = '\0';
    printf("%s\n", buffer);
    return i;
}
int lastrow(char *s, int d) // 上一行
{
    int i, count = 0;
    char *buffer = malloc(100);
    int py = d;
    for (; d > 0; d--)
    {
        if (s[d] == '\n')
            count++;
        if (count == 2)
            break;
    }
    memcpy(buffer, s + d + 1, py - d - 2);
    buffer[py - d - 2] = '\0';
    printf("%s\n", buffer);
    return d;
}

```

这段代码接收一个参数用来指定文件名，随后先使用 `fstat` 函数获得文件的大小，保证后续对文件内容的读取操作不发生越界，然后使用 `mmap` 函数将文件的内容映射到内存中。其中第一个参数 `start` 为 `NULL` 时表示由系统决定映射区的起始地址；第二个参数 `length` 表示映射区的长度，不足一页按一页处理，这里即为前面取得的文件大小；第三个参数期望的内存保护标志 `prot` 的 `PROT_READ` 表示页内容可以被读取；第四个参数映射的

对象的类型 flags 的 MAP\_PRIVATE 表示建立一个写入时拷贝的私有映射。内存区域的写入不会影响到原文件；第五个参数 fd 表示有效的文件描述词，一般是由 open 函数返回；第六个参数 off\_t offset 表示被映射对象内容的起点。mmap 函数的返回值为映射区内存的起始地址，此后调用的三个函数 onepage、nextrow、lastrow 的功能分别是显示一页内容、显示下一行内容、显示上一行内容。输入 p，继续读 10 行；输入 n，显示下一行；输入 l，显示上一行；输入 q，退出。

运行该程序，映射 7.c 的文件内容到内存，使用 p 显示一页（10 行）内容，使用 l 显示上一行内容，使用 n 显示下一行内容，q 退出，效果如下图：



```
hollowman@hollow-virtualbox:~/文档
nano 7.c
gcc 7.c
./a.out 7.c
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <string.h>
#include <memory.h>
#include <stdlib.h>
#include <stdio.h>
int lastrow(char *s, int d);
p
int nextrow(char *s, int d);
int onepage(char *s, int d);
int main(int argc, char *argv[])
{
    if( argc == 2 ) // 判断参数是否只有一个
    {
        int fd, play = 0;
        char lab;
        char *start;
        struct stat sb;
l
        struct stat sb;
n
        struct stat sb;
n
        fd = open(argv[1], O_RDONLY); // 以只读方式打开文件
q
```