操作系统实验报告

题目要求

实现思路

源代码

运行结果

文件读写编程题目

myecho.c

题目要求

- myecho.c的功能与系统echo程序相同
- 接受命令行参数，并将参数打印出来，例子如下：

```
$ ./myecho x
x
$ ./myecho a b c
a b c
```

实现思路

操作系统命令行参数格式：

```
int main(int argc, char *argv[]);
```

`argc`， 命令行参数的个数； `argv`， 命令行参数数组

所以实现 echo， 只需要将 `argv[]` 中除了第一个参数以外的其它参数输出即可。

源代码

```c
#include <stdio.h>

int main(int argc, char* argv[])
{
    int i;
    for (i = 1; i < argc; i++) {
        printf("%s ", argv[i]);
    }
    putchar(10);

    return 0;
}
```

运行结果

```
$ ./myecho x
x
$ ./myecho a bc c
a bc c
```

mycat.c

题目要求

- mycat.c的功能与系统cat程序相同
- mycat将指定的文件内容输出到屏幕，例子如下：
- 要求使用系统调用open/read/write/close实现

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
$ ./mycat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
```

实现思路

1. 从命令行参数获取文件名称
2. 打开文件，读取一个字符，输出显示一个字符，直至文件结束；关闭文件
3. 添加上常见的错误处理

源代码

```c
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    if (argc != 2) {
        perror("Arguments invalid!\n");
        exit(EXIT_FAILURE);
    }

    const char* fname = argv[1];
    int fd = open(fname, O_RDONLY);
    if (fd < 0) {
        perror("Can't open file!\n");
        exit(EXIT_FAILURE);
    }

    char ch;
    while (read(fd, &ch, 1) > 0) {
        putchar(ch);
    }
    close(fd);

    return 0;
}
```

运行结果

```
$ echo 123 > test
$ ./mycat test
123
```

mycp.c

题目要求

- mycp.c的功能与系统cp程序相同
- 将源文件复制到目标文件，例子如下：
- 要求使用系统调用open/read/write/close实现

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
$ ./mycp /etc/passwd passwd.bak
$ cat passwd.bak
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
```

实现思路

1. 获取参数
2. 打开要读的文件，要写到的文件；
3. 从打开的文件读取一个字符并输出到指定文件（类似 mycat 的过程）直至文件结束
4. 关闭文件
5. 添加上常见错误的处理

源代码

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    if (argc != 3) {
        perror("Argument error!\n");
        exit(EXIT_FAILURE);
    }

    const char* fsrc = argv[1];
    const char* fdst = argv[2];

    // open src
    int fd1 = open(fsrc, O_RDONLY);
    if (fd1 < 0) {
        perror("can't open src\n");
        exit(EXIT_FAILURE);
    }

    // create dst
    int fd2 = creat(fdst, 0666);
    if (fd2 < 0) {
        perror("can't open dst\n");
        exit(EXIT_FAILURE);
    }

    // cp
    char ch;
    while (read(fd1, &ch, 1) > 0) {
        write(fd2, &ch, 1);
    }

    // close file
    close(fd1);
    close(fd2);

    return 0;
}
```

运行结果

```
$ ./mycp mycp.c back_cp
$ diff mycp.c back_cp
$
```

多进程题目

mysys.c

题目要求

- mysys的功能与系统函数system相同，要求用进程管理相关系统调用自己实现一遍
- 使用**fork/exec/wait系统调用实现mysys**
- 不能通过调用系统函数system实现mysys
- 测试程序

```c
#include <stdio.h>

void mysys(char *command)
{
    实现该函数，该函数执行一条命令，并等待该命令执行结束
}

int main()
{
    printf("--------------------------------------------------\n");
    mysys("echo HELLO WORLD");
    printf("--------------------------------------------------\n");
    mysys("ls /");
    printf("--------------------------------------------------\n");
    return 0;
}
```

- 测试程序的输出结果

```
--------------------------------------------------
HELLO WORLD
--------------------------------------------------
bin    core  home            lib         mnt   root  snap  tmp  vmlinuz
boot   dev   initrd.img      lost+found  opt   run   srv   usr  vmlinuz.old
cdrom  etc   initrd.img.old  media       proc  sbin  sys   var
--------------------------------------------------
```

实现思路

1. 首先拆分命令 command，将其按照空格拆分成字符串数组 argv
2. fork() 创建子进程, 在子进程中调用 execvp (因为每次调用会清空进程环境，所以采用在子进程中调用系统命令的方法)
3. 主进程 wait()，待子进程调用系统命令结束后，结束函数 mysys()

源代码

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

void split(char* argv[], char* s)
{
    int cnt = 0, k = 0;
    while (s[k]) {
        while (s[k] && s[k] == ' ') k++;
        if (s[k]) argv[cnt++] = s + k;
        while (s[k] && s[k] != ' ') k++;
        if (s[k]) s[k++] = '\0';
    }
    argv[cnt] = NULL;
}

void mysys(const char* command)
{
    char cmd[80], *argv[20];
    strncpy(cmd, command, 80);
    split(argv, cmd);

    pid_t pid = fork();
    if (pid == 0) {
        execvp(argv[0], argv);
        exit(0);
    }
    wait(NULL);
}

int main()
{
    printf("--------------------\n");
    mysys("echo HELLO WORLD");
    printf("--------------------\n");
    mysys("ls /");
    printf("--------------------\n");

    return 0;
}
```

运行结果

```
$ ./mysys
--------------------
HELLO WORLD
--------------------
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
--------------------
```

sh1.c

题目要求

- 该程序读取用户输入的命令，调用函数mysys(上一个作业)执行用户的命令，示例如下

```
# 编译sh1.c
$ cc -o sh1 sh1.c

# 执行sh1
$ ./sh

# sh1打印提示符>，同时读取用户输入的命令echo，并执行输出结果
> echo a b c
a b c

# sh1打印提示符>，同时读取用户输入的命令cat，并执行输出结果
> cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

- 请考虑如何实现内置命令cd、pwd、exit

实现思路

1. 在 mysys.c 的基础上稍作修改即可，在主循环中读入命令，将需要执行的命令交给 mysys 去做
2. exit、cd 命令需要特殊处理，则在调用 mysys 执行前判断，是这两条命令时就做特殊处理
3. 对于 `exit`：直接跳出循环即可；对于 `cd`：可调用系统函数 chdir() 来实现

源代码

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define MAX_SIZE 80

void split(char* argv[], char* s)
{
    int cnt = 0, k = 0;
    while (s[k]) {
        while (s[k] && s[k] == ' ') k++;
        if (s[k]) argv[cnt++] = s + k;
        while (s[k] && s[k] != ' ') k++;
        if (s[k]) s[k++] = '\0';
    }
    argv[cnt] = NULL;
}

void mysys(const char* command)
{
    char cmd[80], *argv[20];
    strncpy(cmd, command, 80);
    split(argv, cmd);

    pid_t pid = fork();
    if (pid == 0) {
        execvp(argv[0], argv);
        exit(0);
    }
    wait(NULL);
}


int main()
{
    char cmd[MAX_SIZE];

    while (1) {
        char* p = fgets(cmd, MAX_SIZE, stdin);
        if (p == NULL) break;

        cmd[strlen(cmd)-1] = '\0';  // '\n' -> '\0'

        if (strcmp(cmd, "exit") == 0) {
            break;
        } else if (strncmp(cmd, "cd", 2) == 0) {
            if (chdir(cmd + 3) < 0)
                puts("directory errer!");
            continue;
        }
```

```
        mysys(cmd);
    }

    return 0;
}
```

运行结果

```
$ ./sh1
ls
mysys  mysys.c  sh1  sh1.c  sh2.c  split.c
pwd
/home/guest/practice/part2
cd ..
ls
part1  part2  part3
pwd
/home/guest/practice
echo a bb ccc
a bb ccc
exit
$
```

sh2.c

题目要求

要求在第1版的基础上，添加如下功能

- 实现文件重定向

```
# 执行sh2
$ ./sh2

# 执行命令echo，并将输出保存到文件log中
> echo hello >log

# 打印cat命令的输出结果
> cat log
hello
```

实现思路

1. 为了实现重定向，需要对 传入的参数进行解析，得到输入文件和输出文件；
2. 具体实现时，修改 split 函数，对 command 结构进行封装，在子进程执行exec前重定向即可

3. 添加的 command 结构：

```
// struct Command
struct Command {
    char *argv[20];                         // 命令行参数
    int in, out;                            // 是否有 输入/输出 重定向
    char *infile, *outfile;             // 重定向的文件名
};
```

源代码

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>

#define MAX_SIZE 80

// deal error
void panic(const char* s)
{
    puts(s);
    exit(EXIT_FAILURE);
}

// struct Command
struct Command {
    char *argv[20];
    int in, out;
    char *infile, *outfile;
};

void split(struct Command* cmd, char* s)
{
    // split by BLANK
    int cnt = 0, k = 0;
    while (s[k]) {
        while (s[k] && s[k] == ' ') k++;
        if (s[k]) cmd->argv[cnt++] = s + k;
        while (s[k] && s[k] != ' ') k++;
        if (s[k]) s[k++] = '\0';
    }
    cmd->argv[cnt] = NULL;
    // set Redirect File
    if (cmd->argv[cnt-1][0] == '<') {
        cmd->in = 1;
        cmd->infile = cmd->argv[cnt-1] + 1;
    } else if (cmd->argv[cnt-1][0] == '>') {
        cmd->out = 1;
        cmd->outfile = cmd->argv[cnt-1] + 1;
    }
    if (cmd->in || cmd->out) cmd->argv[cnt-1] = NULL;
}

void mysys_exec(struct Command* cmd)
{
    pid_t pid;
    pid = fork();

    if (pid == 0) {
        // redirect
```

```
        if (cmd->in) {
            int fd = open(cmd->infile, O_RDONLY);
            if (fd < 0) panic("infile");
            dup2(fd, 0);
        }
        if (cmd->out) {
            int fd = creat(cmd->outfile, 0666);
            if (fd < 0) panic("outfile");
            dup2(fd, 1);
        }
        // run
        execvp(cmd->argv[0], cmd->argv);
        // exit
        exit(0);
    }
    wait(NULL);
}

void mysys(const char* command)
{
    char str[MAX_SIZE];
    strncpy(str, command, MAX_SIZE);
    // split command
    struct Command cmd;
    memset(&cmd, 0, sizeof(cmd));
    split(&cmd, str);
    // run command
    mysys_exec(&cmd);
}


int main()
{
    char cmd[MAX_SIZE];

    while (1) {
        char* p = fgets(cmd, MAX_SIZE, stdin);
        if (p == NULL) break;

        cmd[strlen(cmd)-1] = '\0';  // '\n' -> '\0'

        if (strcmp(cmd, "exit") == 0) {
            break;
        } else if (strncmp(cmd, "cd", 2) == 0) {
            if (chdir(cmd + 3) < 0)
                puts("directory errer!");
            continue;
        }

        mysys(cmd);
    }
```

```
    return 0;
}
```

运行结果

```
guest@box:~/practice/part2$ cc sh2.c -o sh2
guest@box:~/practice/part2$ ./sh2
echo hello >log
cat log
hello
exit
guest@box:~/practice/part2$
```

sh3.c

题目要求

- 实现管道

  ```
  # 执行sh3
  $ ./sh3

  # 执行命令cat和wc，使用管道连接cat和wc
  > cat /etc/passwd | wc -l
  ```

- 考虑如何实现管道和文件重定向

  ```
  $ cat input.txt
  3
  2
  1
  3
  2
  1
  $ cat <input.txt | sort | uniq | cat >output.txt
  $ cat output.txt
  1
  2
  3
  ```

实现思路

需要在 sh2.c 的基础上做多处修改

1. 在 mysys 中可能会执行多个命令（即调用多次 exec，因为命令中会用管道 '|' ），所以需要对命令进行拆分，分别执行

2. 在两个命令执行之间还需要用到管道，所以还需要加上管道的创建，以及将管道连接到子进程的过程
3. 在主进程创建管道且 fork() 后，等待子进程运行结束前，需要本进程的关闭管道连接（否则会在子进程中产生 pipe read 时阻塞的现象）

源代码

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>

#define MAX_SIZE 80
#define MAX_PART 20

// deal error
void panic(const char* s)
{
    puts(s);
    exit(EXIT_FAILURE);
}

// struct Command
struct Command {
    char *argv[20];
    int in, out;
    char *infile, *outfile;
};

void split(struct Command* cmd, char* s, int begin, int end)
{
    // split by BLANK
    int cnt = 0, k = begin;
    while (k < end) {
        while (k < end && s[k] == ' ') k++;
        if (k < end) cmd->argv[cnt++] = s + k;
        while (k < end && s[k] != ' ') k++;
        if (k < end) s[k++] = '\0';
    }
    cmd->argv[cnt] = NULL;
    // set Redirect File
    if (cmd->argv[cnt-1][0] == '<') {
        cmd->in = 1;
        cmd->infile = cmd->argv[cnt-1] + 1;
    } else if (cmd->argv[cnt-1][0] == '>') {
        cmd->out = 1;
        cmd->outfile = cmd->argv[cnt-1] + 1;
    }
    if (cmd->in || cmd->out) cmd->argv[cnt-1] = NULL;
}

void mysys_exec(struct Command* cmd, int fd_in, int fd_out)
{
    pid_t pid;
    pid = fork();

    if (pid == 0) {
```

```
        // redirect
        if (fd_in >= 0) {
            dup2(fd_in, 0);
            close(fd_in);
        }
        if (fd_out >= 0) {
            dup2(fd_out, 1);
            close(fd_out);
        }
        // file redirect
        if (cmd->in) {
            int fd = open(cmd->infile, O_RDONLY);
            if (fd < 0) panic("infile");
            dup2(fd, 0);
            close(fd);
        }
        if (cmd->out) {
            int fd = creat(cmd->outfile, 0666);
            if (fd < 0) panic("outfile");
            dup2(fd, 1);
            close(fd);
        }
       // run
        execvp(cmd->argv[0], cmd->argv);
        // exit
        exit(0);
    }
    // close
    close(fd_in);
    close(fd_out);
    wait(NULL);
}

void mysys(const char* command)
{
    char str[MAX_SIZE];
    struct Command cmds[MAX_PART];
    int fd[MAX_PART][2];        // pipe
    strncpy(str, command, MAX_SIZE);
    memset(cmds, 0, sizeof(cmds));

    // split
    int begin = 0, end, cnt = 0;
    while (str[begin]) {
        for (end = begin; str[end] && str[end] != '|'; end++);
        split(&cmds[cnt++], str, begin, end);
        if (!str[end]) break;
        str[end] = '\0';
        begin = end + 1;
    }
```

```
    int i;
    for (i = 0; i < cnt; i++) {
        // set pipe
        if (i < cnt-1) pipe(fd[i]);
        int fd_in = i == 0? dup(0): fd[i-1][0];
        int fd_out = i == cnt-1? dup(1): fd[i][1];
        // run command
        mysys_exec(&cmds[i], fd_in, fd_out);
    }
}


int main()
{
    char cmd[MAX_SIZE];

    while (1) {
        printf("my$ ");
        char* p = fgets(cmd, MAX_SIZE, stdin);
        if (p == NULL) break;

        cmd[strlen(cmd)-1] = '\0';  // '\n' -> '\0'

        if (strcmp(cmd, "exit") == 0) {
            break;
        } else if (strncmp(cmd, "cd", 2) == 0) {
            if (chdir(cmd + 3) < 0)
                puts("directory errer!");
            continue;
        }

        mysys(cmd);
    }

    return 0;
}
```

运行结果

```
uest@box:~/practice/part2$ cc sh3.c -o sh3
guest@box:~/practice/part2$ cat input.txt
3
2
1
3
2
1
guest@box:~/practice/part2$ ./sh3
my$ cat <input.txt | sort | uniq | cat >output.txt
my$ cat output.txt
1
2
3
my$
```

多线程题目

pi1.c

题目要求

使用2个线程根据莱布尼兹级数计算PI

- 莱布尼兹级数公式: 1 - 1/3 + 1/5 - 1/7 + 1/9 - ... = PI/4
- 主线程创建1个辅助线程
- 主线程计算级数的前半部分
- 辅助线程计算级数的后半部分
- 主线程等待辅助线程运行結束后,将前半部分和后半部分相加

实现思路

按照流程具体实现即可，主要是线程创建及传参接受返回值部分

1. 创建辅助线程
2. 主线程计算前一半
3. 主线程计算完成后使用 `pthread_join` 等待辅助线程的返回值

源代码

```c
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <stdlib.h>

#define TOT_NUM 2000

void *worker_proc(void* arg)
{
    // calculate
    double r = 0, a;
    int i;
    for (i = TOT_NUM/2; i < TOT_NUM; i++) {
        a = 1.0 / (i*2 + 1);
        if (i & 1) a = -a;
        r += a;
    }
    printf("worker: %f\n", r);

    // return value
    double *rst;
    rst = (double*)malloc(sizeof(double));
    *rst = r;
    return (void*)rst;
}

int main()
{
    // create worker
    pthread_t worker;
    pthread_create(&worker, NULL, worker_proc, NULL);

    // main
    double r0 = 0, a;
    int i;
    for (i = 0; i < TOT_NUM/2; i++) {
        a = 1.0 / (i*2 + 1);
        if (i & 1) a = -a;
        r0 += a;
    }
    printf("main: %f\n", r0);

    double r1, r;
    // receive rst
    void* rst;
    pthread_join(worker, (void**)&rst);
    r1 = *(double*)rst;
    free(rst);

    r = (r0 + r1) * 4;
    printf("tot_num: %d\npi: %f\n", TOT_NUM, r);
```

```
    return 0;
}
```

运行结果

```
guest@box:~/practice/part3$ cc pi1.c -lpthread
guest@box:~/practice/part3$ ./a.out
main: 0.785148
worker: 0.000125
tot_num: 2000
pi: 3.141093
guest@box:~/practice/part3$
```

pi2.c

题目要求

实现思路

源代码

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

#define TOT_NUM 2000
#define CORE_NUM 4
#define PART_NUM (TOT_NUM/CORE_NUM)

// thread arg
struct Param {
    int begin;
    int end;
};

// thread result
struct Result {
    double value;
};

void *worker_proc(void *arg)
{
    // get arg
    struct Param *p = (struct Param *)arg;
    int s = p->begin, t = p->end;
    // cal
    int i;
    double r = 0, a;
    for (i = s; i < t; i++) {
        a = 1.0 / (i*2 + 1);    // 1.0 !! double
        if (i & 1) a = -a;      // i not a !!
        r += a;
    }
    // return result
    struct Result *rst;
    rst = (struct Result *)malloc(sizeof(struct Result));
    rst->value = r;
    return rst;
}

int main()
{
    pthread_t workers[CORE_NUM];
    struct Param params[CORE_NUM];

    // create thread
    int i;
    for (i = 0; i < CORE_NUM; i++) {
        struct Param *p;
        p = &params[i];
        p->begin = PART_NUM * i;
        p->end = p->begin + PART_NUM;
```

```
        pthread_create(&workers[i], NULL, worker_proc, p);
    }

    // join thread
    double r = 0;
    for (i = 0; i < CORE_NUM; i++) {
        struct Result *rst;
        pthread_join(workers[i], (void**)&rst);
        r += rst->value;
        free(rst);
    }

    printf("pi: %f\n", r*4);

    return 0;
}
```

运行结果

```
guest@box:~/practice/part3$ cc pi2.c -lpthread
guest@box:~/practice/part3$ ./a.out
pi: 3.141093
```

sort.c

题目要求

实现思路

源代码

```c
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>

#define ARY_SIZE 100

void print(int *a, int n)
{
    int i;
    for (i = 0; i < n; i++) {
        printf("%8d", a[i]);
    }
}

// gen data
void gen(int *a, int s, int t)
{
    int i;
    for (i = s; i < t; i++) {
        a[i] = rand() % 1000;
    }
}

void select_sort(int *a, int s, int t)
{
    int i, j;
    for (i = s; i < t-1; i++) {
        int p = i;
        for (j = i+1; j < t; j++) {
            if (a[p] > a[j]) {
                p = j;
            }
        }
        if (p != i) {
            int t = a[p];
            a[p] = a[i];
            a[i] = t;
        }
    }
}

void merge(int *a, int s, int m, int t)
{
    int b[ARY_SIZE];
    int i = s, j = m, k = 0;

    while (i < m && j < t) {
        if (a[i] < a[j]) b[k++] = a[i++];
        else b[k++] = a[j++];
    }
```

```
    while (i < m) b[k++] = a[i++];
    while (j < t) b[k++] = a[j++];

    memcpy(a, b, ARY_SIZE*sizeof(int));
}

struct Param {
    int *arry;
    int begin;
    int end;
};

// thread proc
void *worker_proc(void *arg)
{
    // get arg
    struct Param *p;
    p = (struct Param *)arg;
    int s = p->begin;
    int t = p->end;
    int *a = p->arry;
    // cal
    select_sort(a, s, t);
}


int main()
{
    // gen data & show
    int a[ARY_SIZE];
    gen(a, 0, ARY_SIZE);
    puts("before sort:----------------");
    print(a, ARY_SIZE);

    // create thread
    pthread_t worker;
    struct Param param;
    param.arry = a;
    param.begin = ARY_SIZE / 2;
    param.end = ARY_SIZE;
    pthread_create(&worker, NULL, worker_proc, &param);

    // main
    select_sort(a, 0, ARY_SIZE / 2);
    // join
    pthread_join(worker, NULL);

    // merge & print
    merge(a, 0, ARY_SIZE/2, ARY_SIZE);
    puts("after sort:-----------------");
    print(a, ARY_SIZE);
```

```
    return 0;
}
```

运行结果

```
guest@box:~/practice/part3$ cc sort.c -lpthread
guest@box:~/practice/part3$ ./a.out
before sort:----------------
383     886     777     915     793     335     386     492     649     421
362      27     690      59     763     926     540     426     172     736
211     368     567     429     782     530     862     123      67     135
929     802      22      58      69     167     393     456      11      42
229     373     421     919     784     537     198     324     315     370
413     526      91     980     956     873     862     170     996     281
305     925      84     327     336     505     846     729     313     857
124     895     582     545     814     367     434     364      43     750
87      808     276     178     788     584     403     651     754     399
932      60     676     368     739      12     226     586      94     539
after sort:-----------------
11      12      22      27      42      43      58      59      60      67
69      84      87      91      94      123     124     135     167     170
172     178     198     211     226     229     276     281     305     313
315     324     327     335     336     362     364     367     368     368
370     373     383     386     393     399     403     413     421     421
426     429     434     456     492     505     526     530     537     539
540     545     567     582     584     586     649     651     676     690
729     736     739     750     754     763     777     782     784     788
793     802     808     814     846     857     862     862     873     886
895     915     919     925     926     929     932     956     980     996
guest@box:~/practice/part3$
```

pc1.c

题目要求

实现思路

源代码

运行结果

pc2.c

题目要求

实现思路

源代码

运行结果

ring.c

题目要求

实现思路

源代码

运行结果