操作系统实验报告

陈学润

2017 年 6 月 22 日

目录

1 文件读写

1.1 myecho

1.1.1 题目

- 实现系统程序echo的功能，将参数打印出来
- 例子

$ ./myecho a
a
$ ./myecho a b c
a b c

1.1.2 程序清单

```
#include<stdio.h>
int main(int argc, char **argv){
        int i;
        for(i=1;i<argc;i++)
        {
                if(i!=1)
                        pritnf(" ");
                printf("%s",argv[i]);
        }
        printf("\n");
}
```

1.1.3 分析

在终端输入 *./myechoabc* 后，操作系统处理完字符串，并把参数argc和argv交给程序，在例子中，

argc=4 argv[0]="myecho" argv[1]="a" argv[2]="b" argv[3]="c"

程序将每个参数输出到屏幕。

1.2 mycat

1.2.1 题目

- 实现系统程序cat的功能，将文件打印出来
- 例子

```
$ cat test.txt
hello world
$ ./mycat test.txt
hello world
```

1.2.2 程序清单

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char **argv){
        int id,flag,i,count;
        count=0;
        char *p;
        char buf[100];
        char *filename=argv[1];
        id=open(filename,O_RDONLY);
        if(id==-1){
                printf("file open error!\n");
                exit(0);
        }
        while(flag=read(id,&buf,100)!=0){
                for(p=buf,i=0;i<flag;i++,p++){
                        putchar(*p);
                        if((*p)=='\n')
                                if(++count==24)
                                        getchar();
                }
        }
```

```
22          return 0;
23  }
```

1.2.3 分析

读取文件，并将文件的每个字符输出到屏幕。

1.3 mycp

1.3.1 题目

- 实现系统程序cp的功能，复制文件
- 例子

$ cat test.txt
hello world
$ ./mycp test.txt test.bak
$ cat test.bak
hello world

1.3.2 程序清单

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<sys/stat.h>
4  #include<fcntl.h>
5  #include<unistd.h>
6  int main(int argc,char **argv){
7          int num,b,c;
8          int inputfileid,outputfileid;
9          char buf[100];
10
11         inputfileid=open(argc[1],O_RDONLY);
12         if(inputfileid==-1){
13                 printf("inputfile open error!\n");
```

```
14                  exit(0);
15          }
16          outputfileid=creat(argv[2],666);
17          if(outputfileid==-1){
18                  printf("outputfile open error!\n");
19                  exit(0);
20          }
21
22          while(num=read(inputfileid,&buf,100)!=0){
23                  write(outputfileid,&buf,num);
24          }
25
26          close(inputfileid);
27          close(outputfileid);
28  }
```

1.3.3 分析

创建两个文件，一个读取源文件，一个写进新文件，不断读取源文件，直到读到文件尾部为止。

2 多进程

2.1 mysys

2.1.1 题目

实现函数mysys，用于执行一个系统命令，要求如下

- 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
- 测试程序如下

```
#include <stdio.h>
int main()
{
    printf("--------------------------------\n");
    mysys("echo HELLO WORLD");
    printf("--------------------------------\n");
    system("ls /");
    mysys("--------------------------------\n");
    return 0;
}
```

2.1.2 程序清单

```
1 void mysys(char input[]){
2
3     char **argv=(char **)malloc(5*sizeof(char *));
4     char *result=NULL;
5     char *p=input;
6     int argc=0,k=0;
7     char delims=' ';
8
```

```
9       result=(char *)malloc(50);
10      while(*p!='\0'){
11          if(*p!=' '){
12              result[k++]=*p;
13              p++;
14          }else{
15              argv[argc++]=result;
16                          result=(char *)malloc(50);
17                          k=0;
18                          p++;
19          }
20      }
21      argv[argc]=NULL;
22
23      int pid;
24      pid=fork();
25      if(pid==0){
26          int succeed=0;
27          if(argc==0){
28              exit(0);
29          }else if(argc==1){
30              succeed=execlp(argv[0],argv[0],NULL);
31              if(succeed==-1){
32                  printf("%s",argv[0]);
33              }
34              exit(0);
35          }
36          succeed=execvp(argv[0],argv);
37          if(succeed==-1){
38              printf("command not exist\n");
39          }
40          exit(0);
41      }
```

```
42
43      wait(NULL);
44  }
```

2.1.3 分析

mysys拿到字符串input[]后，将其分割处理后的结果赋给argv,之后创建子进程，在子进程中调用exec调用argv，实现了system的功能。

2.2 sh1

2.2.1 题目

实现shell程序，要求具备如下功能

- 支持命令参数

  $ echo arg1 arg2 arg3
  $ ls /bin /usr/bin /home

- 实现内置命令cd、pwd、exit

  $ cd /bin
  $ pwd
  /bin

2.2.2 程序清单

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4  #include<unistd.h>
5  #include<sys/types.h>
6  #include<sys/stat.h>
7  #include<sys/wait.h>
8  #include<fcntl.h>
9
```

```
10 #define MAX_L 30
11 #define MAX_P 5
12
13 char cmd[MAX_L];
14 char param[MAX_P][MAX_L];
15 char *para[MAX_P+1];
16
17 void split(char str[]){
18     int i=0;
19     char *p=strtok(str," ");
20     while(p!=NULL){
21         strcpy(param[i++],p);
22         p=strtok(NULL," ");
23     }
24     for(int j=0;j<i;j++){
25         para[j]=param[j];
26     }
27     para[i]=NULL;
28 }
29
30 int main(){
31
32     int pid,res;
33
34     for(;;){
35         fgets(cmd,MAX_L,stdin);
36         cmd[strlen(cmd)-1]='\0';
37         split(cmd);
38
39         if(strcmp(para[0],"exit")==0){
40             break;
41         }else if(strcmp(para[0],"cd")==0){
42             chdir(para[1]);
```

```
43                continue;
44            }
45
46            pid=fork();
47
48            if(pid<0){
49                perror("Create process fail!\n");
50                exit(1);
51            }else  if(pid==0){
52                res=execvp(para[0],para);
53                if(res==-1){
54                    perror("Execute fail!\n");
55                    exit(1);
56                }
57            }else{
58                wait(NULL);
59            }
60        }
61    }
```

2.2.3 分析

相比mysys，改进了字符串分割操作，增加了split()函数。

cd操作用chdir()实现。

fgets(stdin)会把最后一个换行符也读进来，忽略这一点给调试带来了很大的麻烦。

2.3 sh2

2.3.1 题目

实现shell程序，要求在第1版的基础上，添加如下功能

- 实现文件重定向

$ echo hello ¿log

```
$ cat log
hello
```

2.3.2 程序清单

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<fcntl.h>

#define MAX_L 30
#define MAX_P 5

char cmd[MAX_L];
char param[MAX_P][MAX_L];
char *para[MAX_P+1];

int split(char str[]){
    int i=0;
    char *p=strtok(str," ");
    while(p!=NULL){
        strcpy(param[i++],p);
        p=strtok(NULL," ");
    }
    for(int j=0;j<i;j++){
        para[j]=param[j];
    }
    para[i]=NULL;
    return i;
}
```

```
30
31 void outRedirect(char **pa){
32     char f[30];
33     int fd;
34     strcpy(f,*pa+1);
35     *pa=NULL;
36
37     fd=creat(f,666);
38     if(fd==-1){
39         perror("Create new file fail.\n");
40         exit(1);
41     }
42     dup2(fd,STDOUT_FILENO);
43     close(fd);
44 }
45
46 int main(){
47
48     int pid,res,paramsize;
49
50     for(;;){
51         fgets(cmd,MAXL,stdin);
52         cmd[strlen(cmd)-1]='\0';
53         paramsize=split(cmd);
54
55         if(strcmp(para[0],"exit")==0){
56             break;
57         }else if(strcmp(para[0],"cd")==0){
58             chdir(para[1]);
59             continue;
60         }
61
62         pid=fork();
```

```
63
64          if(pid<0){
65              perror("Create process fail!\n");
66              exit(1);
67          }else  if(pid==0){
68
69              if(para[paramsize-1][0]=='>'){
70                  outRedirect(&para[paramsize-1]);
71              }
72
73              res=execvp(para[0],para);
74              if(res==-1){
75                  perror("Execute fail!\n");
76                  exit(1);
77              }
78          }else{
79              wait(NULL);
80          }
81      }
82  }
```

2.3.3 分析

相比mysys，多了实现了一个文件重定向函数outRedirect(),该函数将argv最后一个字符串取出并根据其内容创建文件，把标准输出重定向到改文件，之后把参数值置为NULL。

2.4 sh3

2.4.1 题目

实现shell程序，要求在第2版的基础上，添加如下功能

- 实现管道

$ cat /etc/passwd — wc -l

- 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
```

2.4.2 程序清单

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<unistd.h>
4  #include<string.h>
5  #include<sys/types.h>
6  #include<sys/stat.h>
7  #include<sys/wait.h>
8  #include<fcntl.h>
9
10 #define MAX_L 100
11 #define MAX_P 10
12
13 char cmd[MAX_L];
14 char param[MAX_P][MAX_L];
15 char *para[MAX_P+1];
16 int subcmd[MAX_P];
17 int paramsize;
```

```
18 int subcmdnum;
19
20 void split(char str[]){
21
22     int scn=0;
23     int ps=0;
24     subcmd[scn++]=ps;
25     char *p=strtok(str," ");
26     while(p!=NULL){
27         strcpy(param[ps],p);
28
29         if(strcmp(p,"|")==0){
30             para[ps]=NULL;
31             subcmd[scn++]=ps+1;
32         }else{
33             para[ps]=param[ps];
34         }
35
36         ps++;
37         p=strtok(NULL," ");
38     }
39     para[ps]=NULL;
40     subcmdnum=scn;
41     paramsize=ps;
42 }
43
44 void restore(){
45         for(int i=0;i<subcmdnum;i++){
46                 for(int j=0;(para+subcmd[i])[j]!=NULL;j++){
47                         printf("%s ",(para+subcmd[i])[j]);
48                 }
49                 if(i!=subcmdnum-1){
50                         printf("| ");
```

```
51                      }
52              }
53
54  }
55
56  void outRedirect(char **pa){
57      char f[30];
58      int fd;
59      strcpy(f,*pa+1);
60      *pa=NULL;
61
62      fd=creat(f,444);
63      if(fd==-1){
64          perror("Create new file fail.\n");
65          exit(1);
66      }
67      dup2(fd,STDOUT_FILENO);
68      close(fd);
69  }
70
71  void tes(char *a[]){
72      int i=0;
73      while(a[i]!=NULL){
74          printf("%s ",a[i++]);
75      }
76      printf("\n");
77  }
78
79  void main(){
80      int pid,fstdin,fstdout,res,count;
81      int pfd[4][2];
82
83      fstdin=dup(STDIN_FILENO);
```

```
84          fstdout=dup(STDOUT_FILENO);
85
86      for(;;){
87
88                  fgets(cmd,MAX_L,stdin);
89          cmd[strlen(cmd)-1]='\0';
90          split(cmd);
91                  //restore();
92
93          if(strcmp(para[0],"exit")==0){
94              break;
95          }else  if(strcmp(para[0],"cd")==0){
96              chdir(para[1]);
97              continue;
98          }
99
100         count=0;
101
102         while(count<subcmdnum-1){
103                         res=pipe(pfd[count]);
104                         if(res!=0){
105                         perror("Create pipe fail!\n");
106                         exit(1);
107                 }
108
109             pid=fork();
110             if(pid<0){
111                 printf("Create process fail! count=%d\n",count);
112                 exit(1);
113             }else  if(pid==0){
114                 dup2(pfd[count][1],STDOUT_FILENO);
115                 close(pfd[count][0]);
116                 close(pfd[count][1]);
```

```
117                 res=execvp(para[subcmd[count]],para+subcmd[count]);
118                                 if(res==-1){
119                                         printf("child:Execute fail! count=
120                                         exit(1);
121                                 }
122                 exit(0);
123             }else{
124                         dup2(pfd[count][0],STDIN_FILENO);
125                 close(pfd[count][0]);
126                 close(pfd[count][1]);
127                                 //printf("father:count=%d\n",count);
128                                 count++;
129             }
130         }
131
132         pid=fork();
133         if(pid<0){
134             perror("Create process fail!\n");
135             exit(1);
136         }else  if(pid==0){
137             if(para[paramsize-1][0]=='>'){
138                                 //printf("%s\n",para[paramsize-1]);
139                 outRedirect(&para[paramsize-1]);
140             }
141             res=execvp(para[subcmd[count]],para+subcmd[count]);
142             if(res==-1){
143                 printf("child:Execute fail! count=%d\n",count);
144                 exit(1);
145             }
146                         exit(0);
147         }else{
148                         //printf("father:count=%d\n",count);
149             wait(NULL);
```

```
150             }
151             dup2(fstdin ,STDIN_FILENO);
152                     dup2(fstdout ,STDOUT_FILENO);
153         }
154 }
```

2.4.3 分析

restore()和tes()作用为显示命令，在调试时使用，程序运行时无用。outRedirect()无改变。

split()函数有许多变化，这在于，不仅需要将命令按空格划分成数组，还要按照管道的操作划分命令，记录下管道操作的数目count，并记录下每个管道操作命令在数组中的起始下标，这样方便了之后调用execvp()函数。

在主函数中，申请了一个二维数组，保存了4个管道，fstdin和fstdout用于在循环中将被重定向的标准输入输出定向回来，res用于保存各种函数调用的返回值，count用于管道计数。

进入外层循环，每次循环输入一个命令，然后split()处理该字符串，若为exit则结束循环退出程序。按count的值进行内层循环，直到到达通道命令个数subcmdnum，每次循环都会开启一个管道，让子进程的标准输出重定向到管道写端，父进程的标准输入重定向到管道的读端。在下一轮循环的子进程，读取上一轮子进程传来的数据，调用execvp后结果传给这一轮的父进程，如此往复，直到通道循环结束，把最后一个结果输出到文件。

3 多线程

3.1 pi1

3.1.1 题目

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

3.1.2 程序清单

```
1  //func.c
2  #include<stdlib.h>
3
4  float element(int n){
5      float result;
6      if(n%2==1){
7          result=1.0/(2*n-1);
8      }else{
9          result=-1.0/(2*n-1);
10     }
11 //     printf("n=%d,result=%f\n",n,result);
12     return result;
13 }
14
15 float cacu(int first,int last){
16     float sum=0;
17     for(int i=first;i<=last;i++){
18         sum+=element(i);
```

```
19      }
20  //      printf("first=%d,last=%d,sum=%f\n",first ,last ,sum);
21      return sum;
22  }
23
24  void *thr(void *arg){
25      int *arr=(int *)arg;
26      float *result=malloc(sizeof(float));
27      *result=cacu(arr[0]+1,arr[1]);
28      return result;
29  }
```

```
1   //pi1.c
2   #include <pthread.h>
3   #include <stdlib.h>
4   #include <stdio.h>
5
6   #define N 50
7
8   float element(int n);
9   float cacu(int,int);
10  void *thr(void *);
11
12  int main(){
13      pthread_t pth;
14      int arg[]={N/2,N};
15      float sum,sum1,*sum2;
16      int fail=pthread_create(&pth,NULL,&thr,arg);
17      if(fail){
18          printf("fail to create thread\n");
19          exit(0);
20      }
21      sum1=cacu(1,N/2);
```

```
22      pthread_join(pth,(void **)&sum2);
23      sum=sum1+(*sum2);
24      printf("sum=%f,sum1=%f,sum2=%f\n",sum,sum1,*sum2);
25  }
```

3.1.3 分析

主线程计算前一半数之和存到sum1,子线程计算后一半数之和存到sum2，两者相加。

3.2 pi2

3.2.1 题目

使用N个线程根据莱布尼兹级数计算PI

- 与上一题类似，但本题更加通用化，能适应N个核心，需要使用线程参数来实现
- 主线程创建N个辅助线程
- 每个辅助线程计算一部分任务，并将结果返回
- 主线程等待N个辅助线程运行结束，将所有辅助线程的结果累加

3.2.2 程序清单

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #define N 1000
6  #define CPU_N 10
7
8  float element(int);
9  float cacu(int,int);
10 void *thr(void *);
```

```
11
12 int main(){
13     pthread_t CPU[CPU_N];
14     int arg[CPU_N];
15     float sum=0;
16
17     for(int i=0;i<CPU_N;i++){
18         arg[i]=i*N/CPU_N;
19     }
20
21     for(int i=0;i<CPU_N;i++){
22         int fail=pthread_create(&CPU[i],NULL,&thr,&arg[i]);
23         if(fail){
24             printf("fail to create thread\n");
25             exit(0);
26         }
27     }
28
29     for(int i=0;i<CPU_N;i++){
30         float *result;
31         pthread_join(CPU[i],(void **)&result);
32         printf("CPU[%d] %f\n",i+1,*result);
33         sum=sum+(*result);
34     }
35
36     printf("sum=%f\n",sum);
37 }
```

3.2.3 分析

子函数已经编好，主函数传值即可，与pi1无异。

3.3 sort

3.3.1 题目

多线程排序

- 主线程创建一个辅助线程
- 主线程使用选择排序算法对数组的前半部分排序
- 辅助线程使用选择排序算法对数组的后半部分排序
- 主线程等待辅助线程运行結束后,使用归并排序算法归并数组的前半部分和后半部分

3.3.2 程序清单

```
1  //sortfunc.c
2  #include<stdlib.h>
3  #include<time.h>
4  #include<stdio.h>
5
6  void check(int *arr,int start,int end){
7      int right=1;
8      for(int i=start;i<end;i++){
9          if(arr[i]>arr[i+1]){
10             printf("arr[%d]=%d,arr[%d]=%d\n",i,arr[i],i+1,arr[i+1]);
11             right=0;
12             break;
13         }
14     }
15     if(right){
16         printf("right\n");
17     }
18 }
19
20 void selectionSort(int *arr,int start,int end){
```

```
21      printf("start␣selectionSort␣from␣%d␣to␣%d\n",start,end);
22          int min,i,j,temp;
23          for(i=start;i<end;i++){
24                  min=i;
25                  for(j=i+1;j<=end;j++){
26                          if(arr[min]>arr[j])
27                                  min=j;
28                  }
29                  if(min!=i){
30                          temp=arr[i];
31                          arr[i]=arr[min];
32                          arr[min]=temp;
33                  }
34          }
35      check(arr,start,end);
36      printf("%d-%d␣completed\n",start,end);
37  }
38
39  void merge(int *arr,int *temp,int start,int mid,int end){
40      printf("start␣merge␣from␣%d␣to␣%d\n",start,end);
41          int i=start,j=mid+1,k=start;
42          while(i!=mid+1&&j!=end+1){
43                  if(arr[i]>arr[j])
44                          temp[k++]=arr[j++];
45                  else
46                          temp[k++]=arr[i++];
47          }
48          while(i!=mid+1)
49                  temp[k++]=arr[i++];
50          while(j!=end+1)
51                  temp[k++]=arr[j++];
52          for(i=start;i<=end;i++)
53                  arr[i]=temp[i];
```

```
54      check(arr,start,end);
55      printf("%d-%d completed\n",start,end);
56  }
57
58  void init(int *arr,int n){
59      srand(time(NULL));
60      for(int i=0;i<n;i++){
61          arr[i]=random();
62      }
63  }
```

```
1   //sort.c
2   #include<stdio.h>
3   #include<pthread.h>
4   #include<stdlib.h>
5
6   #define N 1000
7
8   typedef struct{
9           int *arr;
10          int start;
11          int end;
12  }str_arr;
13
14  void merge(int *,int *,int,int,int);
15  void selectionSort(int *,int,int);
16  void init(int *,int);
17  void check(int *,int,int);
18
19  void *chr(void *arg){
20          str_arr *in=(str_arr *)arg;
21          selectionSort(in->arr,in->start,in->end);
22  }
```

```
23
24 int main(){
25         int array[N];
26         int temp[N];
27         init(array,N);
28
29         str_arr input;
30         input.arr=array;
31         input.start=0;
32         input.end=N/2-1;
33
34         pthread_t pth;
35         pthread_create(&pth,NULL,&chr,&input);
36         selectionSort(array,N/2,N-1);
37         pthread_join(pth,NULL);
38
39         merge(array,temp,0,N/2-1,N-1);
40
41 }
```

3.3.3 分析

将数组排序信息input传给子线程，即可与主线程一起排序，最后合并。

3.4 pc1

3.4.1 题目

使用条件变量解决生产者、计算者、消费者问题

- 系统中有3个线程：生产者、计算者、消费者
- 系统中有2个容量为4的缓冲区：buffer1、buffer2
- 生产者生产'a'、'b'、'c'、'd'、'e'、'f'、'g'、'h'八个字符，放入到buffer1

- 计算者从buffer1取出字符，将小写字符转换为大写字符，放入到buffer2
- 消费者从buffer2取出字符，将其打印到屏幕上

3.4.2 程序清单

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

#define CAP 4
#define TABLE_SIZE 8
#define ITEM_COUNT 1000

char table[]={'a','b','c','d','e','f','g','h'};

pthread_mutex_t mutex1;
pthread_mutex_t mutex2;
pthread_cond_t wait_empty_buffer1;
pthread_cond_t wait_full_buffer1;
pthread_cond_t wait_empty_buffer2;
pthread_cond_t wait_full_buffer2;

typedef struct{
    char buf[CAP+1];
    int head;
    int tail;
}buffer;

buffer b1,b2;

int buffer_full(buffer *b){
    if((b->tail+CAP)%(CAP+1)==b->head)
```

```
29          return 1;
30      else
31          return 0;
32  }
33
34  int buffer_empty(buffer *b){
35      if(b->tail==b->head)
36          return 1;
37      else
38          return 0;
39  }
40
41  void produce(buffer *b,char c){
42      b->buf[b->head]=c;
43      b->head=(b->head+1)%(CAP+1);
44      printf("tail=%d,head=%d\n",b->tail,b->head);
45  }
46
47  void count(buffer *b1,buffer *b2){
48  //      char temp=b1->buf[b1->tail]-32;
49      b2->buf[b2->head]=b1->buf[b1->tail]-32;
50      b2->head=(b2->head+1)%(CAP+1);
51      b1->tail=(b1->tail+1)%(CAP+1);
52  }
53
54  char comsume(buffer *b){
55      char c=b->buf[b->tail];
56  //      printf("comsumer print:%c\n",c);
57      b->tail=(b->tail+1)%(CAP+1);
58      printf("tail=%d,head=%d\n",b->tail,b->head);
59      return c;
60  }
61
```

```
62 void *pro_thr(void *arg){
63     for(int i=0;i<ITEM_COUNT;i++)
64     {
65         pthread_mutex_lock(&mutex1);
66 
67         while(buffer_full(&b1)){
68             pthread_cond_wait(&wait_empty_buffer1,&mutex1);
69         }
70         char c=table[rand()%TABLE_SIZE];
71         produce(&b1,c);
72         pthread_cond_signal(&wait_full_buffer1);
73         pthread_mutex_unlock(&mutex1);
74     }
75 }
76 
77 void *cou_thr(void *arg){
78     for(int i=0;i<ITEM_COUNT;i++){
79         pthread_mutex_lock(&mutex1);
80 
81         while(buffer_empty(&b1)){
82             pthread_cond_wait(&wait_full_buffer1,&mutex1);
83         }
84 
85         char c=comsume(&b1);
86         pthread_cond_signal(&wait_empty_buffer1);
87         pthread_mutex_unlock(&mutex1);
88 
89         c=c-32;
90         pthread_mutex_lock(&mutex2);
91 
92         while(buffer_full(&b2)){
93             pthread_cond_wait(&wait_empty_buffer2,&mutex2);
94         }
```

```
95
96          produce(&b2,c);
97          pthread_cond_signal(&wait_full_buffer2);
98          pthread_mutex_unlock(&mutex2);
99      }
100 }
101
102 void *com_thr(void *arg){
103     for(int i=0;i<ITEM_COUNT;i++){
104         pthread_mutex_lock(&mutex2);
105
106         while(buffer_empty(&b2)){
107             pthread_cond_wait(&wait_full_buffer2,&mutex2);
108         }
109         char c=comsume(&b2);
110         pthread_cond_signal(&wait_empty_buffer2);
111         pthread_mutex_unlock(&mutex2);
112     }
113 }
114
115 int main(){
116     srand(time(NULL));
117     pthread_t producer,counter,comsumer;
118     pthread_mutex_init(&mutex1,NULL);
119     pthread_mutex_init(&mutex2,NULL);
120     pthread_cond_init(&wait_empty_buffer1,NULL);
121     pthread_cond_init(&wait_empty_buffer2,NULL);
122     pthread_cond_init(&wait_full_buffer1,NULL);
123     pthread_cond_init(&wait_full_buffer2,NULL);
124
125     b1.head=b1.tail=b2.head=b2.tail=0;
126
127     pthread_create(&producer,NULL,pro_thr,NULL);
```

```
128     pthread_create(&counter,NULL,cou_thr,NULL);
129     pthread_create(&comsumer,NULL,com_thr,NULL);
130
131     pthread_join(producer,NULL);
132     pthread_join(counter,NULL);
133     pthread_join(comsumer,NULL);
134
135     printf("ok\n");
136     return 0;
137 }
```

3.4.3 分析

使用两个队列b1,b2作为两个缓冲区。创建三个线程分别代表生产者，计算者，消费者。

对两个缓冲区加互斥锁mutex1,mutex2，对两个缓冲区的'空和满'加条件变量来同步，他们是

wait_empty_buffer1 wait_full_buffer1

wait_empty_buffer2 wait_full_buffer2

当某线程准备往已满的缓冲区添加时，阻塞在wait_empty_buffer

当某线程取走缓冲区里的东西时，唤醒对应的wait_empty_buffer

当某线程准备向空缓冲区取东西时，阻塞在wait_full_buffer

当某线程向缓冲区添加了东西时，唤醒对应的wait_full_buffer

3.5 pc2

3.5.1 题目

使用信号量解决生产者、计算者、消费者问题

- 功能和前面的实验相同，使用信号量解决

3.5.2 程序清单

```
1 #include<stdio.h>
2 #include<pthread.h>
```

```
3  #include<stdlib.h>
4  #include<time.h>
5
6  #define CAP 4
7  #define TABLE_SIZE 8
8  #define ITEM_COUNT 1000
9
10 char table[]={'a','b','c','d','e','f','g','h'};
11
12 typedef struct{
13     int value;
14     pthread_mutex_t mutex;
15     pthread_cond_t cond;
16 }sema_t;
17
18 void sema_init(sema_t *sema,int value){
19     sema->value=value;
20     pthread_mutex_init(&sema->mutex,NULL);
21     pthread_cond_init(&sema->cond,NULL);
22 }
23
24 void sema_wait(sema_t *sema){
25     pthread_mutex_lock(&sema->mutex);
26     while(sema->value<=0)
27         pthread_cond_wait(&sema->cond,&sema->mutex);
28     sema->value--;
29     pthread_mutex_unlock(&sema->mutex);
30 }
31
32 void sema_signal(sema_t *sema){
33     pthread_mutex_lock(&sema->mutex);
34     sema->value++;
35     pthread_cond_signal(&sema->cond);
```

```
36      pthread_mutex_unlock(&sema->mutex);
37  }
38
39  sema_t mutex_b1,mutex_b2;
40  sema_t empty_b1,full_b1;
41  sema_t empty_b2,full_b2;
42
43  typedef struct{
44      char buf[CAP+1];
45      int head;
46      int tail;
47  }buffer;
48
49  buffer b1,b2;
50
51  void produce(buffer *b,char c){
52      b->buf[b->head]=c;
53      b->head=(b->head+1)%(CAP+1);
54      printf("tail=%d,head=%d\n",b->tail,b->head);
55  }
56
57  char comsume(buffer *b){
58      char c=b->buf[b->tail];
59      b->tail=(b->tail+1)%(CAP+1);
60      printf("tail=%d,head=%d\n",b->tail,b->head);
61      return c;
62  }
63
64  void *pro_thr(void *arg){
65      for(int i=0;i<ITEM_COUNT;i++)
66      {
67          char c=table[rand()%TABLE_SIZE];
68
```

```
69          sema_wait(&empty_b1);
70          sema_wait(&mutex_b1);
71          produce(&b1,c);
72          printf("producer: %d %c\n",i+1,c);
73          sema_signal(&full_b1);
74          sema_signal(&mutex_b1);
75      }
76  }
77
78  void *cou_thr(void *arg){
79      for(int i=0;i<ITEM_COUNT;i++){
80          sema_wait(&full_b1);
81          sema_wait(&mutex_b1);
82          char c=comsume(&b1);
83          sema_signal(&empty_b1);
84          sema_signal(&mutex_b1);
85
86          c=c-32;
87
88          sema_wait(&empty_b2);
89          sema_wait(&mutex_b2);
90          produce(&b2,c);
91          printf("counter: %d %c\n",i+1,c);
92          sema_signal(&full_b2);
93          sema_signal(&mutex_b2);
94      }
95  }
96
97  void *com_thr(void *arg){
98      for(int i=0;i<ITEM_COUNT;i++){
99          sema_wait(&full_b2);
100         sema_wait(&mutex_b2);
101         char c=comsume(&b2);
```

```
102             printf("comsumer: %d %c\n",i+1,c);
103             sema_signal(&empty_b2);
104             sema_signal(&mutex_b2);
105         }
106 }
107
108 int main(){
109     srand(time(NULL));
110     pthread_t producer,counter,comsumer;
111     sema_init(&mutex_b1,1);
112     sema_init(&mutex_b2,1);
113     sema_init(&empty_b1,4);
114     sema_init(&empty_b2,4);
115     sema_init(&full_b1,0);
116     sema_init(&full_b2,0);
117
118     b1.head=b1.tail=b2.head=b2.tail=0;
119
120     pthread_create(&producer,NULL,pro_thr,NULL);
121     pthread_create(&counter,NULL,cou_thr,NULL);
122     pthread_create(&comsumer,NULL,com_thr,NULL);
123
124     pthread_join(comsumer,NULL);
125
126     printf("ok\n");
127     return 0;
128
129 }
```

3.5.3 分析

对两个缓冲区的条件变量进行改造，成为信号量，极大的简化了代码。

3.6 ring

3.6.1 题目

创建N个线程，它们构成一个环

- 创建N个线程：T1、T2、T3、… TN
- T1向T2发送整数1
- T2收到后将整数加1
- T2向T3发送整数2
- T3收到后将整数加1
- T3向T4发送整数3
- …
- TN收到后将整数加1
- TN向T1发送整数N

3.6.2 程序清单

```
1  #include<stdio.h>
2  #include<pthread.h>
3  #include<stdlib.h>
4
5  #define N 10
6
7  int buf[N];
8  pthread_t thr[N];
9  //pthread_cond_t buffull[N];
10 //pthread_mutex_t bufmut[N];
11 int stop=0;
12
13 typedef struct{
14     int value;
```

```
15      pthread_mutex_t mutex;
16      pthread_cond_t cond;
17  }sema_t;
18
19  sema_t semas[N];
20
21  void sema_init(sema_t *sema,int value){
22      sema->value=value;
23      pthread_mutex_init(&sema->mutex,NULL);
24      pthread_cond_init(&sema->cond,NULL);
25  }
26
27  void sema_wait(sema_t *sema){
28      pthread_mutex_lock(&sema->mutex);
29      while(sema->value<=0)
30          pthread_cond_wait(&sema->cond,&sema->mutex);
31      sema->value--;
32      pthread_mutex_unlock(&sema->mutex);
33  }
34
35  void sema_signal(sema_t *sema){
36      pthread_mutex_lock(&sema->mutex);
37      sema->value++;
38      pthread_cond_signal(&sema->cond);
39      pthread_mutex_unlock(&sema->mutex);
40  }
41
42  void *first(void *arg){
43      int *index=(int *)(arg);
44      int temp=1;
45      while(!stop){
46          buf[*index+1]=temp;
47          printf("%d %d\n",*index,temp);
```

```
48          //pthread_cond_signal(&buffull[*index+1]);
49          sema_signal(&semas[*index+1]);
50          sema_wait(&semas[*index]);
51          //pthread_cond_wait(&buffull[*index],&bufmut[*index]);
52          temp=buf[*index]+1;
53      }
54  }
55
56  void *other(void *arg){
57      int *index=(int *)(arg);
58      int t=(*index==N-1?0:*index+1);
59      while(!stop){
60          //pthread_cond_wait(&buffull[*index],&bufmut[*index]);
61          sema_wait(&semas[*index]);
62          buf[t]=buf[*index]+1;
63          printf("%d %d\n",*index,buf[t]);
64          sema_signal(&semas[t]);
65          //pthread_cond_signal(&buffull[t]);
66      }
67  }
68
69  int main(){
70      int zero=0,digit[N];
71      for(int i=0;i<N;i++){
72          sema_init(&semas[i],0);
73      }
74      pthread_create(&thr[0],NULL,first,(void *)(&zero));
75      for(int i=1;i<N;i++){
76          digit[i]=i;
77          pthread_create(&thr[i],NULL,other,(void *)(&digit[i]));
78      }
79      printf(".....\n");
80      getchar();
```

```
81      stop=1;
82      for(int i=0;i<N;i++)
83          pthread_join(thr[i],NULL);
84      return 0;
85  }
```

3.6.3 分析

创建一个长度为N的数组，用于保存每个线程发送的数据，再创建一个长度N的信号量数组，分别对应每个缓冲区的资源，取值为1表示有新的数据，取值为0表示没有新的数据，第一个线程先发数据再收数据，其他线程与其相反，所以编写了两个线程函数。