南京航空航天大学

计算机操作系统实验报告

实验： 操作系统实验.

姓　　名： 李应飞
学　　号： 161610338
指导老师： 朱广蔚
日　　期： 2019.6.28

目录

一、文件读写编程题目

1.myecho.c

1.1 问题：

(1)myecho.c 的功能与系统 echo 程序相同

(2)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

1.2 算法思想：

因为 argc 是运行程序送给 main 函数的命令行参数个数，argv 是指针数组，所以可以利用循环直接输出指针数组 argv 里的值。

1.3 源代码：

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
    int i=1;
        while(i < argc)
        {
                printf("%s ",argv[i]);
                i++;
        }
    printf("\n");
        return 0;
}
```

2.mycat.c

2.1 问题

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
...
```

2.2 算法思想

利用 open 函数打开传来的参数（文件名），lseek 函数定位读的位置，并利用 offset 作为偏移量。每次读指定大小的数据，当读的数据长度小于指定长度时，读完成。

2.3 源代码

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

int main(int argc,char *argv[])
{
        int i = 1;
        while(i < argc)
        {
                int fp    = open(argv[i], O_RDONLY);//打开文件，只读方式
                if(fp == -1)
                {
                        printf("open error!\n");
                        exit(0);
                }
                char buf[100];
                int offset=0;
                int word=1;
                while(word != 0)
                {
                   lseek(fp,offset,SEEK_SET);//定位读的位置
                   memset(buf,0,100);//清空字符数组
                   int t1=read(fp , buf, 90);//读数据，返回值为数据长度
                   if(t1 == -1)
                   {
                      printf("read error!\n");
                      word=0;
                   }
                   else
                   {
                      printf("%s",buf);
                      if(t1 < 90)//计算偏移量
```

```
                        word=0;
                    offset+=t1;
                }
            }
        close(fp);
        printf("\n");
        i++;
    }
}
```

3.mycp.c

3.1 问题

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
...
```

3.2 算法思想

打开源文件与目标文件，一个只读方式，一个只写方式。把源文件的内容全部写到目标文件中。

3.3 源代码

```
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
int main(int argc,char *argv[])
{
        int fp1 = open(argv[1],O_RDONLY);
        if(fp1 == -1)
        {
                printf("open1 error!\n");
```

```
        exit(0);
    }
    int fp2 = creat(argv[2],00644);//创建文件，后面为文件的操作权限
    if(fp2 == -1)
    {
        printf("open2 error!\n");
        exit(0);
    }
    char buf[100];
    memset(buf , 0 , 100);
    int t1, offset=0;
    int word=1;
    while(word!=0)
    {
        lseek(fp1,offset,SEEK_SET);//定位文件
        memset(buf,0,100);//清空字符数组
        t1=read(fp1,buf,90);//读
        if(t1 == -1)
        {
            printf("file1 read error!\n");
            word=0;

        }
        else
        {
            int t2=write(fp2,buf,90);//写
            if(t2 == -1)
            {
                printf("file2 write error!\n");
                exit(0);
            }

            if(t1  < 90)//计算偏移量
                word=0;
            offset+=t1;
        }
    }
    close(fp1);
    close(fp2);
    printf("\n");
    return 0;
}
```

二、多进程题目

1.mysys.c

1.1 问题

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

```
#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
```

2.2 算法思想

split 函数中利用 strtok 函数来分割字符串 command，得到分割长度 len 和指针数组 argv，len 用于检验字符串是否分割正确。然后创建子进程，利用 execvp 函数装入程序（argv 指定传递给程序的参数，argv 数组的最后一项必须是 NULL 指针）。最后执行。

2.3 源代码

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

```
#include <sys/wait.h>
#define Max 100

void split(char *command,int *len,char *argv[Max])//分割字符串
{
        int i=0;
        char *p;
//因为 strtok 函数会改变字符串数组结构，所以 com 用于存 command
        char *com=(char *)malloc(sizeof(char) * strlen(command) + 1);//
开辟空间
        strcpy(com,command);
        p=strtok(com," ");
//      printf("**\n");
        while(p != NULL)
        {
                argv[i]=(char *)malloc(sizeof(char) * 100);
                strcpy(argv[i],p);
                i++;
//              printf("%s\n",argv[i-1]);
//              printf("%s\n",p);
                p=strtok(NULL," ");
        }
//      printf("here!!\n");
/*      int k=0;
        while(k < i)
        {
                printf("k = %d\n", k);
                printf("%s\n",argv[k]);
                k++;
        }*/
        *len=i;
        argv[i]=(char *)malloc(sizeof(char)*100);
        argv[i]=NULL;//最后置空，后面装入程序需要

//      printf("i = %d",i);
}
void mysys(char *command)
{
        char *argv[Max];
        int len;
        split(command,&len,argv);//分割字符串
/**     printf("after split\n");
        int j=0;
        printf("len = %d\n", len);
```

```
        while(j<len)
        {
                printf("j: %d\n", j);
                printf("%s\n",argv[j]);
                j++;
        }
*/
//      argv[i]=NULL;
        pid_t pid;//创建子进程
        pid=fork();
        if (pid == 0)
        {
                int error=execvp(argv[0],argv);//装入程序
                if(error < 0)
                {
                        perror("execvp");
                }
        }
        wait(NULL);//等待进程结束
}

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

2.sh1.c

2.1 问题

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

2.2 算法思想

split 函数中利用 strtok 函数来分割字符串 command，得到分割长度 len 和字符数组 *argv，len 用于检验字符串是否分割正确。mysys 函数中在创建 Judgy 函数判断命令是 cd、pwd、exit、cat、echo 哪一种。如果是 exit，直接退出程序；如果是 cd，利用 chdir 函数来实现；如果是 pwd，利用 getcwd 函数来实现，其余返回，然后创建子进程，利用 execvp 函数装入程序(argv 指定传递给程序的参数,argv 数组的最后一项必须是 NULL 指针）。最后执行。

2.3 源代码

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <malloc.h>
#include <sys/wait.h>

#define Max 100
#define MAX 1000
void split(char  *command,int *len,char *argv[Max])//分割字符串
{
    int i=0;
    char *p;
//因为 strtok 函数会改变字符串数组结构，所以 com 用于存 command
    char *com=(char *)malloc(sizeof(char)*strlen(command) +1);
    strcpy(com,command);
    p=strtok(com," ");
    while(p !=NULL)
    {
        argv[i]=(char *)malloc(sizeof(char)*100);
        strcpy(argv[i],p);
        i++;
        p=strtok(NULL," ");
    }
    *len=i;
```

```
    argv[i]=(char *)malloc(sizeof(char)*100);
    argv[i]=NULL;
}
int Judgy(char **argv)//判断是哪一种命令
{
    if(strcmp(argv[0],"exit") == 0)
    {
        exit(0);
    }
    else if(strcmp(argv[0],"cd") == 0)
    {
        if(chdir(argv[1]))
        {
            printf("cd:%s no such directory\n",argv[1]);
        }
        return 1;
    }
    else if(strcmp(argv[0],"pwd") == 0)
    {
        char buf[MAX];
        printf("%s\n",getcwd(buf,sizeof(buf)));
        return 1;
    }
    return 0;
}
void mysys(char *command)
{
    char *argv[Max];
    int len;
    split(command,&len,argv);

//    int j=0;
    printf("len=%d\n",len);
/*    while(j<len)
    {
        printf("%s\n",argv[j]);
        j++;
    }*/
    if(Judgy(argv)) return;
    pid_t pid;//创建子进程进程
    pid=fork();
    if(pid == 0)
    {
        int error=execvp(argv[0],argv);
```

```
        if(error < 0)
        {
            perror("execvp");
        }
    }
    wait(NULL);
}

int main(int argc,char *argv[])
{
//   extern void mysys(char *command);
   char ch[MAX];
   memset(ch,0,MAX);
   printf(">");
   gets(ch);
   while(ch!=NULL)
   {
    // if(!strcmp(ch,"exit")) break;
     mysys(ch);
     printf(">");
     gets(ch);
  // printf("ch=%s",ch);
    // mysys(ch);
   }
//   printf("\n");
  // mysys("ls /");
    return 0;
}
```

3.sh2.c

3.1 问题

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

```
# 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
```

3.2 算法思想

增加 loog 函数，其中利用 dup2 函数实现重定向，即将输出保存到文件 log 中。可以参考教程例子。

3.3 源代码

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

#define Max 100
#define MAX 1000
void split(char  *command,int *len,char *argv[Max])//分割字符串
{
    int i=0;
    char *p;
    char *com=(char *)malloc(sizeof(char)*strlen(command) +1);
    strcpy(com,command);
    p=strtok(com," ");
    while(p !=NULL)
    {
        argv[i]=(char *)malloc(sizeof(char)*100);
        strcpy(argv[i],p);
        i++;
        p=strtok(NULL," ");
    }
    *len=i;
    argv[i]=(char *)malloc(sizeof(char)*100);
    argv[i]=NULL;
}
int Judgy(char **argv)//判断是哪一种命令
{
    if(strcmp(argv[0],"exit") == 0)
    {
        exit(0);
    }
    else if(strcmp(argv[0],"cd") == 0)
    {
        if(chdir(argv[1]))
        {
            printf("cd:%s no such directory\n",argv[1]);
        }
```

```
            return 1;
        }
        else if(strcmp(argv[0],"pwd") == 0)
        {
            char buf[MAX];
            printf("%s\n",getcwd(buf,sizeof(buf)));
            return 1;
        }
        return 0;
    }
    int loog(char **argv,int len)//重定向
    {
        if(strcmp(argv[0],"echo") == 0)
        {
            if(*(*(argv+(len-1))+0) == '>')//判断最后一个字符串是不是含有>
            {
                char *p;
                p=(char *)malloc(sizeof(char)*100);//字符串 p 用于存文件名
                int length=strlen(argv[len-1]);
                int j=1,i=0;
                while(j<length)
                {
                    p[i]=*(*(argv+len-1)+j);
                    j++;
                    i++;
                }
                p[i]='\0';

                int fd;
                fd = open(p,O_APPEND|O_CREAT|O_RDWR,0666);//打开 log 文件
                dup2(fd,1);// 首先关闭文件描述符 1，然后把文件描述符 1 指向文
件描述 fd
                close(fd);// 最后把文件描述符 fd 关闭
                argv[len-1]=NULL;//将>log 置空,方便装入程序
            }

        }
    }
    void mysys(char *command)
    {
        char *argv[Max];
        int len;
        split(command,&len,argv);//分割字符串
```

```
/*    int j=0;
    printf("len=%d\n",len);
    while(j<len)
    {
        printf("%s\n",argv[j]);
        j++;
    }*/

    if(Judgy(argv)) return;//判断是哪一种命令
    pid_t pid;//创建子进程
    pid=fork();
    if(pid == 0)
    {
        loog(argv,len);//重定向
        int error=execvp(argv[0],argv);//装入程序
        if(error < 0)
        {
            perror("execvp");
        }
    }
    wait(NULL);
}

int main(int argc,char *argv[])
{
   char ch[MAX];
   memset(ch,0,MAX);
   printf(">");
   gets(ch);
   while(ch!=NULL)
   {
    // if(!strcmp(ch,"exit")) break;
     mysys(ch);
     printf(">");
     gets(ch);
   }
    return 0;
}
```

4.sh3.c

4.1 问题

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

```
# 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
```

4.2 算法思想

cat /etc/passwd | wc -l：

Pip（）判断|的位置，返回值为|的位置，用于判断是否是管道命令。Pip1（）通过|分割命令为指针数组 argv 和指针数组 argv1 。利用 dup2（）重定向。

cat <input.txt | sort | uniq | cat >output.txt：

思考：

先判断是否是管道命令，然后分割命令，在通过循环分别实现每个命令。可以利用多个进程，并不是仅有一个进程。

4.3 源代码

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

#define Max 100
#define MAX 1000
void split(char  *command,int *len,char *argv[Max])//分割字符串
{
    int i=0;
    char *p;
    char *com=(char *)malloc(sizeof(char)*strlen(command) +1);
    strcpy(com,command);
```

```
    p=strtok(com," ");
    while(p !=NULL)
    {
        argv[i]=(char *)malloc(sizeof(char)*100);
        strcpy(argv[i],p);
        i++;
        p=strtok(NULL," ");
    }
    *len=i;
    argv[i]=(char *)malloc(sizeof(char)*100);
    argv[i]=NULL;
}
int Judgy(char **argv)//判断命令是什么
{
    if(strcmp(argv[0],"exit") == 0)
    {
        exit(0);
    }
    else if(strcmp(argv[0],"cd") == 0)
    {
        if(chdir(argv[1]))
        {
            printf("cd:%s no such directory\n",argv[1]);
        }
        return 1;
    }
    else if(strcmp(argv[0],"pwd") == 0)
    {
        char buf[MAX];
        printf("%s\n",getcwd(buf,sizeof(buf)));
        return 1;
    }
    return 0;
}
int loog(char **argv,int len)//重定向文件
{
    if(strcmp(argv[0],"echo") == 0)
    {
       // printf("****1\n");
       // printf("len=%d\n",len);
       // printf("%s\n",*(argv+(len-1)));
       // printf("%c\n",*(*(argv+(len-1))+0));
       // if(strcmp((*(argv+(len-1))+0) , ">") == 0)
        if(*(*(argv+(len-1))+0) == '>')
```

```
        {
            char *p;
            p=(char *)malloc(sizeof(char)*100);
            int length=strlen(argv[len-1]);
            int j=1,i=0;
            while(j<length)
            {
                p[i]=*(*(argv+len-1)+j);
                j++;
                i++;
            }
            p[i]='\0';

            int fd;
            fd = open(p,O_APPEND|O_CREAT|O_RDWR,0666);
            dup2(fd,1);
            close(fd);
            argv[len-1]=NULL;
        }

    }
}
int Pip(char **argv,int len)//判断|的位置
{
    int i=0;
    while(i < len)
    {
        if(strcmp(argv[i],"|") == 0)
        {
            return i;
        }
        i++;
    }
    return 0;
}
void Pip1(char **argv,char **argv1,int word,int len)//分割命令管道
{
    int i=word+1,j=0;
    while(i < len)
    {
        argv1[j]=(char *)malloc (sizeof(char )*100);
        strcpy(argv1[j],argv[i]);
        i++;
        j++;
```

```
    }
    argv[word]=NULL;
    argv1[j]=NULL;
}
void mysys(char *command)
{
    char *argv[Max];
    int len;
    split(command,&len,argv);

/*    int j=0;
    printf("len=%d\n",len);
    while(j<len)
    {
        printf("%s\n",argv[j]);
        j++;
    }*/

    if(Judgy(argv)) return;

    int fd[2];//管道
    char buf[32];
    pid_t pid;

    pipe(fd);
    pid=fork();//创建进程

    int word=Pip(argv,len);//判断|的位置
//    printf("| location:%d\n",word);

    if(!word)//word == 0 代表不是管道连接的命令
    {
        if(pid == 0)
        {
            loog(argv,len);
            int error=execvp(argv[0],argv);
            if(error<0)
            {
                perror("execvp");
            }
        }
        wait(NULL);
        return;
    }
```

```
    char **argv1;
    Pip1(argv,argv1,word,len);//分割命令，这里只实现了管道连接两个命令。
  /*  int i=0,j=0;
    while(i < word)
    {
        printf("%s\n",argv[i]);
        i++;
    }
    while(j < (len-word-1))
    {
        printf("%s\n",argv1[j]);
        j++;
    }*/

    if(pid == 0)
    {
        dup2(fd[1],1);//标准输出
        close(fd[0]);//关闭读
        close(fd[1]);//关闭写

        loog(argv,len);
        int error=execvp(argv[0],argv);
        if(error < 0)
        {
            perror("execvp");
        }
//        exit(0);
    }
    dup2(fd[0],0);//标准输入
    close(fd[0]);//关闭读
    close(fd[1]);//关闭写

    int error1=execvp(argv1[0],argv1);
    if(error1 < 0)
    {
        perror("execvp1");
    }
    wait(NULL);
}

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

```
    memset(ch,0,MAX);
    printf(">");
    gets(ch);
    while(ch!=NULL)
    {
     // if(!strcmp(ch,"exit")) break;
      mysys(ch);
      printf(">");
      gets(ch);
    }
     return 0;
}
```

三、多线程题目

1.pi1.c

1.1 问题

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

1.2 算法思想

利用 pthread_create 创建一个辅助线程计算级数的后半部分（即 worker 函数）

利用pthread_join使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，得到结果

1.3 源代码

```
#include <stdio.h>
#include <pthread.h>
#define Max 10000

float worker_output;//辅助线程结果

void *worker(void *arg)//辅助线程计算
{
    int i=(Max/2+1);
    while(i <= Max)
    {
        if(i % 2 == 1)
```

```
        {
            worker_output += 1.0/(2*i-1);
            i++;
        }
        if(i % 2 == 0)
        {
            worker_output -= 1.0/(2*i-1);
            i++;
        }
    }
    return NULL;
}

float master_output;//主线程计算结果

void master()//主线程计算
{
    int i=1;
    while(i <= (Max/2))
    {
        if(i % 2 == 1)
        {
            master_output += 1.0/(2*i-1);
            i++;
        }
        if(i % 2 == 0)
        {
            master_output -= 1.0/(2*i-1);
            i++;
        }
    }
}

int main()
{
    pthread_t worker_tid;
    float  pi;

    pthread_create(&worker_tid,NULL,worker,NULL);//创建辅助线程
    master();

    pthread_join(worker_tid,NULL);//等待辅助线程结束
    pi =4 * (worker_output + master_output);
```

```
printf("worker_output=%f,master_output=%f,pi=%f\n",worker_output,master_output,pi);
        return 0;
    }
```

2.pi2.c

2.1 问题

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

2.2 算法思想

主线程利用 pthread_create 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

利用 pthread_join 使主线程等待 N 个辅助线程运行结束,将所有辅助线程的结果累加

结构体 param 是每个辅助线程的开始计算点到结束计算点

结构体 result 储存结果

2.3 源代码

```
#include <stdio.h>
#include <pthread.h>
#include <malloc.h>
#define Max 10000
#define N_Max 1000
//存放辅助线程的开始与结束点
struct param {
    int start;
    int end;
};
//存放辅助线程结果
struct result{
    float sum;
};

void *worker(void *arg)//辅助进程
{
    struct param *param;
    struct result *result;
    param=(struct param *)arg;
```

```
    int i=param->start;

    result=malloc(sizeof(struct result));
    while(i <= param->end)
    {
        if(i % 2 == 1)
        {
            result->sum += 1.0/(2*i-1);
            i++;
        }
        if(i % 2 == 0)
        {
            result->sum -= 1.0/(2*i-1);
            i++;
        }
    }
    return result;
}

int main()
{
    int i=1;

    int N;
    printf("please input N:");
    scanf("%d",&N);
    int k=Max/N;

    pthread_t worker_tid[N_Max];
    struct param params[N_Max];

    while(i <= N)//创建辅助线程
    {
        struct param *param;
        param=&params[i];
        param->start=(i-1)*k + 1;
        param->end=i*k;

        pthread_create(&worker_tid[i],NULL,worker,param);
        i++;
    }
    float pi;
    int j=1;
```

```
    while(j <= N)//计算结果
    {
        struct result *result;
        pthread_join(worker_tid[j],(void **)&result);
        pi += result->sum;
        free(result);
        j++;
    }
    pi =4 * pi;
    printf("pi=%f\n",pi);
    return 0;
}
```

3.sort.c

3.1 问题

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

3.2 算法思想

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（sort 函数）

主线程等待辅助线程运行結束后,使用归并排序算法（merge 函数）归并数组的前半部分和后半部分

3.3 源代码

```
#include <stdio.h>
#include <pthread.h>
#include <malloc.h>
#include <unistd.h>

#define MAX 100
#define LEN 10

int array[LEN]={0,3,8,6,2,9,5,4,1,7};
struct param {
    int *arr;

};
```

```
void *sort(void *arg)//辅助线程，选择排序
{
    struct param *param;
    param=(struct param *)arg;
    int i=LEN/2,j=0,min=0,temp=0;

    while(i < (LEN -1))
    {
        min=i;
        j=i;
        while(j < LEN)
        {
            if(param->arr[min] > param->arr[j])
                min=j;
            j++;
        }
        temp=param->arr[i];
        param->arr[i]=param->arr[min];
        param->arr[min]=temp;

        i++;
    }
}
void merge()//归并排序
{
    int i=0;
    int a[LEN/2];
    int b[MAX];

    while(i < LEN/2)
    {
        a[i]=array[i];
        i++;
    }

    int j=LEN/2,k=0,count=0;

    while(j < LEN)
    {
        b[k]=array[j];
        j++;
        k++;
        count++;
```

```
    }

    int tm=0,ti=0,tj=0;
    while(ti < LEN/2 && tj < count)
    {
        if(a[ti] < b[tj])
        {
            array[tm]=a[ti];
            ti++;
        }
        else
        {
            array[tm]=b[tj];
            tj++;
        }
        tm++;
    }

    while(tj < count)
    {
        array[tm]=b[tj];
        tj++;
        tm++;
    }
}

int main()
{
    struct param pa;

    pa.arr=array;

    int ti=0,tj=0,tmin=0;
    while(ti < LEN/2)
    {
        tj=ti;
        tmin=ti;
//        printf("%d ",array[ti]);
        while(tj < LEN/2)
        {
            if(array[tmin] > array[tj])
                tmin=tj;
            tj++;
        }
```

```
            int temp=array[tmin];
            array[tmin]=array[ti];
            array[ti]=temp;
            ti++;
        }
      //  printf("\n");
    /*    int s=0;
        while(s < LEN)
        {
            printf("%d ",array[s]);
            s++;
        }
        printf("\n");
    */
        pthread_t worker_id;//创建线程
        pthread_create(&worker_id,NULL,&sort,&pa);
        pthread_join(worker_id,NULL);

    /*    int s=0;
        while(s < LEN)
        {
            printf("%d ",array[s]);
            s++;
        }
        printf("\n");*/

        merge();//归并排序

        int i=0;
        while(i < LEN)
        {
            printf("%d ",array[i]);
            i++;
        }
        printf("\n");
        return 0;
    }
```

4.pc1.c

4.1 问题

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

4.2 算法思想

设置两个容量为 4 的缓冲区：buffer1、buffer2，第一个为生产者和计算者共享，第二个为计算者和消费者共享。，in1 和 out1 作为缓冲区 buffer1 空和满的判断依据，，in2 和 out2 作为缓冲区 buffer2 空和满的判断依据。生产者产生字符，当 buffer1 满时挂起，等待 buffer1 空闲。并且唤醒计算者线程；计算机线程从 buffer1 读字符，对字符操作，并写入 buffer2.满时挂起，并且唤醒消费者线程；消费者线程读取 buffer2 中的字符。

4.3 源代码

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

#define CAPACITY 4 //缓冲区长度
int buffer1[4];//缓冲区 1
int buffer2[4];//缓冲区 2
int in1;//缓冲区 1 指针
int out1;
int in2;//缓冲区 2 指针
int out2;

int buffer1_is_empty()
{
    return in1 == out1;
}

int buffer2_is_empty()
{
    return in2  == out2;
}

int buffer1_is_full()
{
    return (in1 + 1) % CAPACITY == out1;
}

int buffer2_is_full()
{
    return (in2 + 1) % CAPACITY == out2;
}
```

```
int get_item1()
{
    int item1;

    item1 = buffer1[out1];
    out1 = (out1 + 1) % CAPACITY;
    return item1;
}

int get_item2()
{
    int item2;

    item2 = buffer2[out2];
    out2 = (out2 + 1) % CAPACITY;
    return item2;
}

void put_item1(int item)
{
    buffer1[in1] = item;
    in1 = (in1 + 1) % CAPACITY;
}

void put_item2(int item1)
{
    buffer2[in2] = item1;
    in2 = (in2 + 1) % CAPACITY;
}

pthread_mutex_t mutex1;//互斥量
pthread_cond_t wait_empty_buffer1;//条件变量
pthread_cond_t wait_full_buffer1;

pthread_mutex_t mutex2;
pthread_cond_t wait_empty_buffer2;
pthread_cond_t wait_full_buffer2;

#define ITEM_COUNT (CAPACITY * 2)

void *consumer(void *arg)//消费者
{
    int i=0;
    int item;
```

```
    while(i < ITEM_COUNT)
    {
        pthread_mutex_lock(&mutex2);
        while(buffer2_is_empty())
            pthread_cond_wait(&wait_full_buffer2,&mutex2);

        item=get_item2();

        printf("        consume item:%c\n",item);

        pthread_cond_signal(&wait_empty_buffer2);
        pthread_mutex_unlock(&mutex2);
        i++;
    }
    return NULL;
}
void *computer(void *arg)//计算者
{
    int i=0;
    int item;

    while(i < ITEM_COUNT)
    {
        pthread_mutex_lock(&mutex1);
        while(buffer1_is_empty())
        {
            pthread_cond_wait(&wait_full_buffer1,&mutex1);
        }
        item=get_item1();

        printf("    computer get item:%c\n",item);
        item -= 32;

        pthread_cond_signal(&wait_empty_buffer1);
        pthread_mutex_unlock(&mutex1);

        pthread_mutex_lock(&mutex2);
        while(buffer2_is_full())
        {
            pthread_cond_wait(&wait_empty_buffer2,&mutex2);
        }

        put_item2(item);
```

```
        printf("    computer put item:%c\n",item);

        pthread_cond_signal(&wait_full_buffer2);
        pthread_mutex_unlock(&mutex2);

        i++;
    }
    return NULL;
}

void *create(void *arg)//生产者
{
    int i=0;
    int item;

   // printf("*****1\n");
    while(i < ITEM_COUNT)
    {
     //   printf("i=%d\n",i);
        pthread_mutex_lock(&mutex1);
        while(buffer1_is_full())
        {
            pthread_cond_wait(&wait_empty_buffer1,&mutex1);
        }

        item = 'a'+i;
        put_item1(item);

        printf("creat item:%c\n",item);

        pthread_cond_signal(&wait_full_buffer1);
        pthread_mutex_unlock(&mutex1);

        i++;
    }
    return NULL;
}

int main()
{
    pthread_t consumer_tid;
    pthread_t computer_tid;

    pthread_mutex_init(&mutex1,NULL);
```

```
    pthread_mutex_init(&mutex2,NULL);

    pthread_cond_init(&wait_empty_buffer1,NULL);
    pthread_cond_init(&wait_full_buffer1,NULL);

    pthread_cond_init(&wait_empty_buffer2,NULL);
    pthread_cond_init(&wait_full_buffer2,NULL);

    pthread_create(&consumer_tid,NULL,consumer,NULL);
    pthread_create(&computer_tid,NULL,computer,NULL);

   // printf("****\n");
    create(NULL);

    pthread_join(consumer_tid,NULL);
    pthread_join(computer_tid,NULL);

    pthread_mutex_destroy(&mutex1);
    pthread_mutex_destroy(&mutex2);

    return 0;
}
```

5.pc2.c

5.1 问题

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

5.2 算法思想

利用结构体将互斥量、条件变量封装起来作为信号量，并构建信号量 init()，wait()、signal()三个函数。

5.3 源代码

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

#define CAPACITY 4
int buffer1[4];
int buffer2[4];
int in1;
int out1;
int in2;
int out2;
```

```
int buffer1_is_empty()
{
    return in1 == out1;
}

int buffer2_is_empty()
{
    return in2  == out2;
}

int buffer1_is_full()
{
    return (in1 + 1) % CAPACITY == out1;
}

int buffer2_is_full()
{
    return (in2 + 1) % CAPACITY == out2;
}

int get_item1()
{
    int item1;

    item1 = buffer1[out1];
    out1 = (out1 + 1) % CAPACITY;
    return item1;
}

int get_item2()
{
    int item2;

    item2 = buffer2[out2];
    out2 = (out2 + 1) % CAPACITY;
    return item2;
}

void put_item1(int item)
{
    buffer1[in1] = item;
    in1 = (in1 + 1) % CAPACITY;
}
```

```
void put_item2(int item1)
{
    buffer2[in2] = item1;
    in2 = (in2 + 1) % CAPACITY;
}
//信号量结构体
typedef struct{
    int value;
    pthread_mutex_t mutex;
    pthread_cond_t cond;
}sema_t;

void sema_init(sema_t *sema,int value)//初始化信号量
{
    sema->value=value;
    pthread_mutex_init(&sema->mutex,NULL);
    pthread_cond_init(&sema->cond,NULL);
}

void sema_wait(sema_t *sema)//wait () 操作
{
    pthread_mutex_lock(&sema->mutex);
    sema->value--;
    while(sema->value < 0)
    {
        pthread_cond_wait(&sema->cond,&sema->mutex);
    }
    pthread_mutex_unlock(&sema->mutex);
}

void sema_signal(sema_t *sema)//signal () 操作
{
    pthread_mutex_lock(&sema->mutex);
    ++sema->value;
    pthread_cond_signal(&sema->cond);
    pthread_mutex_unlock(&sema->mutex);
}
//定义信号量
sema_t mutex_sema1;
sema_t empty_buffer1_sema;
sema_t full_buffer1_sema;

sema_t mutex_sema2;
```

```
sema_t empty_buffer2_sema;
sema_t full_buffer2_sema;

#define ITEM_COUNT (CAPACITY * 2)

void *consumer(void *arg)///消费者
{
    int i=0;
    int item;

    while(i < ITEM_COUNT)
    {
        sema_wait(&full_buffer2_sema);
        sema_wait(&mutex_sema2);

        item=get_item2();

        sema_signal(&mutex_sema2);
        sema_signal(&empty_buffer2_sema);
        printf("        consume item:%c\n",item);
        i++;
    }
    return NULL;
}
void *computer(void *arg)//计算者
{
    int i=0;
    int item;

    while(i < ITEM_COUNT)
    {
        sema_wait(&full_buffer1_sema);
        sema_wait(&mutex_sema1);

        item=get_item1();

        printf("    computer get item:%c\n",item);

        sema_signal(&mutex_sema1);
        sema_signal(&empty_buffer1_sema);

        sema_wait(&empty_buffer2_sema);
        sema_wait(&mutex_sema2);
```

```
        item -=32;

        put_item2(item);
        printf("    computer put item:%c\n",item);

        sema_signal(&mutex_sema2);
        sema_signal(&full_buffer2_sema);

        i++;
    }
    return NULL;
}

void *create(void *arg)//生产者
{
    int i=0;
    int item;


    while(i < ITEM_COUNT)
    {
        sema_wait(&empty_buffer1_sema);
        sema_wait(&mutex_sema1);

        item = 'a'+i;
        put_item1(item);

        printf("creat item:%c\n",item);

        sema_signal(&mutex_sema1);
        sema_signal(&full_buffer1_sema);

        i++;
    }
    return NULL;
}

int main()
{
    pthread_t consumer_tid;
    pthread_t computer_tid;

    sema_init(&mutex_sema1,1);
    sema_init(&empty_buffer1_sema,CAPACITY-1);
```

```
    sema_init(&full_buffer1_sema,0);

    sema_init(&mutex_sema2,1);
    sema_init(&empty_buffer2_sema,CAPACITY-1);
    sema_init(&full_buffer2_sema,0);

    pthread_create(&consumer_tid,NULL,consumer,NULL);
    pthread_create(&computer_tid,NULL,computer,NULL);

    create(NULL);

    pthread_join(consumer_tid,NULL);
    pthread_join(computer_tid,NULL);


    return 0;
}
```

6.ring.c

6.1 问题

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

6.2 算法思想

利用循环创建 N 个线程，每个线程传参为其标号即 j[i]=i,利用 buffer 与参数的比较实现阻塞与激活线程的条件变量。当相等时，buffer++，激活所有线程。不相等，线程阻塞。

6.3 源代码

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>

#define N 10

int buffer=0;共享
```

```
pthread_mutex_t mutex;
pthread_cond_t wait_cond;

void *T(void *arg)
{
    int i;
    int item;

    int *param=(int *)arg;

    pthread_mutex_lock(&mutex);
    while((*param) != buffer)
    {
        pthread_cond_wait(&wait_cond,&mutex);//阻塞当前线程
    }

    printf("pthread:T%d PUT:%d\n",(*param)+1,(*param)+1);
    buffer = (buffer + 1) % N;
    pthread_cond_broadcast(&wait_cond);//唤醒阻塞在条件变量上的所有线程
    pthread_mutex_unlock(&mutex);
}

int main()
{
    pthread_t pid[N];
    int j[N];
    int i=0;
    pthread_mutex_init(&mutex,NULL);
    pthread_cond_init(&wait_cond,NULL);

    while(i < N)
    {
        j[i]=i;
        pthread_create(&pid[i],NULL,T,(void *)&j[i]);
        i++;
    }

    i=0;
    while(i < N)
    {
        pthread_join(pid[i],NULL);
        i++;
    }
```

```
    return 0;
}
```