1.实验题目

Myecho.c 接受命令行参数并打印出来

实现思路

Main 函数接受 argc 和 argv 参数，分别是参数个数和指向参数内容的二级指针。然后循环打印指针指向的内容。

运行结果

[karl@iZbncwqakj4ds9Z jobs]$ ./a.out x y z

./a.out

x

y

z

源代码

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

2.实验题目

mycat.c 将指定文件内容输出到屏幕。

实现思路

Main 函数接受文件名参数，调用 open 函数，然后循环调用 read 函数读取文件内容，直到 read 函数返回值不大于 0。

运行结果

[karl@iZbncwqakj4ds9Z jobs]$ ./a.out myecho.c

./a.out

#include<stdio.h>

int main(int argc, char **argv){

int i = 0;

for(i=0; i<argc; i++){

printf("%s\n", argv[i]);

}

}

源代码

```
#include<sys/stat.h>
#include<fcntl.h>
#include<stdio.h>
int main(int argc, char **argv){
    int size = 4096;
    char buf[size];
    int fd;
    int index;
```

```
    printf("%s\n", argv[0]);
    fd = open(argv[1], O_RDONLY, 0777);
    while(index = read (fd, buf, size) > 0){
        printf("%s", buf);
        if(index < 0)
            printf("read error\n");
    }
    close(fd);
}
```

3.实验题目

mycp.c 将源文件复制到目标文件

实现思路

使用 open 函数以 0777 权限打开源文件，根据参数创建目标文件，循环调用 read,write 将源文件内容拷贝到目标文件，直到 read 返回值不大于 0。

运行结果

[karl@iZbncwqakj4ds9Z jobs]$ ./a.out myecho.c cp.txt

[karl@iZbncwqakj4ds9Z jobs]$ cat cp.txt

#include<stdio.h>

int main(int argc, char **argv){

int i = 0;

for(i=0; i<argc; i++){

printf("%s\n", argv[i]);

}

}

源代码

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
int main(int argc, char **argv){
    int fd1, fd2;
    int size = 10;
    char buf[size];
    int n;
    fd1 = open(argv[1], O_RDONLY, 0777);
    fd2 = creat(argv[2], 0777);
    while((n = read(fd1, buf, size)) > 0){
        if(write(fd2, buf, n) != n){
            printf("write error\n");
            exit(0);
        }
        if(n < 0){
```

```
            printf("read error\n");
        }
    }
    close(fd1);
    close(fd2);
}
```

4.实验题目

mysys.c 实现 system()函数功能，用 fork/exec/wait 实现 mysys

实现思路

实现 split 函数，该函数可以将一行命令分割成参数数组和参数数量返回。

实现 mysys 函数，该函数先将一行命令用 split 处理，然后将处理结果作为参数，调用 execvp 函数执行。

运行结果

[karl@iZbncwqakj4ds9Z jobs]$ ./a.out

HELLO WORLD

---

alidata boot data etc lib lost+found mnt proc run srv tmp var

bin composer.json dev home lib64 media opt root sbin sys usr

源代码

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

static int command_size = 10;
void split(char *command, int *argc, char **argv){
    int count = 0;
    if(command == NULL || strlen(command) == 0)
        return;
    char *pNext = strtok(command, " ");
    while(pNext != NULL){
        *argv++ = pNext;
        ++count;
        pNext = strtok(NULL, " ");
    }
    *argv = NULL;       //for execvp use
    *argc = count;
}
void mysys(char *com){
    char *command = malloc(sizeof(char)*strlen(com));
    strcpy(command, com);
```

```
    int wordc = 0;
    char *wordv[command_size];
    split(command, &wordc, wordv);
    int pid = fork();
if(pid == 0){
int error = execvp(wordv[0], wordv);
     if(error < 0)
         perror("execvp");
exit(0);
}
wait(NULL);
}
int main(){
printf("----------------------------------------\n");
mysys("echo HELLO WORLD");
printf("----------------------------------------\n");
mysys("ls /");
printf("----------------------------------------\n");
return 0;
}
```

5.实验题目

Sh3.c 实现 shell 功能。

实现 echo 和 cat 命令。

实现内置命令 cd,pwd,exit。

实现文件重定向。

实现管道。

实现多层管道和重定向。

实现思路

用管道符将一条命令分割从多个小命令，每个命令用 cmd 结构体存储，包含 argc,argv,input,output，其中 input 和 output 表示输入和输出重定向。

各条子命令的输入为管道前一条子命令的输出，输出为管道后一条子命令的输入。

运行结果

[karl@iZbncwqakj4ds9Z jobs]$ ./a.out

/home/karl/os/jobs > cd ..

/home/karl/os > cd jobs

/home/karl/os/jobs > pwd

/home/karl/os/jobs

/home/karl/os/jobs > ls

a.out input.txt mycat.c mycp.c myecho.c mysys.c sh4.c

/home/karl/os/jobs > echo x y z >log

/home/karl/os/jobs > cat log

x y z

/home/karl/os/jobs > cat log > cp.txt

```
/home/karl/os/jobs > cat cp.txt
x y z
/home/karl/os/jobs > cat /etc/passwd | wc -l
32
/home/karl/os/jobs > cat input.txt
3
2
1
1
3
2
/home/karl/os/jobs > cat <input.txt | sort | uniq | cat >output.txt
/home/karl/os/jobs > cat output.txt
1
2
3
/home/karl/os/jobs > exit
[karl@iZbncwqakj4ds9Z jobs]$
```

源代码

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

#define MAX_ARGS 8
#define MAX_COMMAND 8
static char *command = NULL;
static int line_size = 256;
static int fdin, fdout, flag;    // flag 为 0 表示管道分割出的第一条命令，为 1 表示最后一条，为 2 表示中间
static int fd[2], fd_temp[2], recover[2];
typedef struct cmd{
    int argc;
    char *argv[MAX_ARGS];
    char *input;
    char *output;
}cmd;
void print_parse(cmd *com);
void readline(char *prompt){
    printf("%s", prompt);
    command = (char *)malloc(sizeof(char)*line_size);
    char ch;
```

```
    int count = 0;
    while((ch = getchar()) != '\n'){
        *(command + count) = ch;
        count += 1;
    }
    *(command + count) = '\0';
}
// 用' '分割出一条命令的指令和参数
void split(char *command, int *argc, char **argv){
    int count = 0;
    if(command == NULL || strlen(command) == 0)
        return;
    char *pNext = strtok(command, " ");
    while(pNext != NULL){
        *argv++ = pNext;
        ++count;
        pNext = strtok(NULL, " ");
    }
    *argv = NULL;
    *argc = count;
}
// 字符串到结构体的处理
void str2cmd(char *str, cmd *p){
    int i;
    char *pNext, *pos;
    if(str == NULL || strlen(str) == 0)
        return;
    if(!strstr(str, ">") && !strstr(str, "<")){
        p->input = NULL;
        p->output = NULL;
    }
    else{
        if(strstr(str, "<")){
            pos = strchr(str, '<');
            pos[0] = '\0';
            for(i = 1;;i++)
                if(pos[i] != ' '){
                    pNext = pos+i;
                    break;
                }
            p->input = strtok(pNext, " ");
            // 为下面做准备，这里容易出错
            for(i=0;;i++)
                if(pNext[i] == '\0'){
```

```
                    pNext += i;
                    break;
                }
            pNext += 1;
        }
        if(strstr(str, ">") || strstr(pNext, ">")){
            if(strstr(str, ">")){
                pos = strchr(str, '>');
            }
            else{
                pos = strchr(pNext, '>');
            }
            pos[0] = '\0';
            for(i = 1;;i++)
                if(pos[i] != ' '){
                    pNext = pos+i;
                    break;
                }
            p->output = strtok(pNext, " ");
        }
    }
    split(str, &p->argc, p->argv);
    //print_parse(p);
}
// 使用 strtok 有未知 bug，只好自己写一个
// 将 pNext 中第一个分割出来的字符串拷到 p，并将 pNext 后移
void my_strtok(char **p, char **pNext, char ch){        // 使用二级指针才能保存修改
    char *temp = NULL;
    int i;
    for(i = 0; (*pNext)[i] != '\0' ;i++){
        if((*pNext)[i] == ch){
            temp = (*pNext)+i+1;
            (*pNext)[i] = '\0';
            break;              // 这里忘记了加 break
        }
    }
    *p = *pNext;

    if(temp == NULL){
        *pNext = NULL;
        return;
    }
    else{
```

```
        for(i = 0;;i++)          // 忽略空格
            if(temp[i] != ' ')
                break;
        temp += i;

        *pNext = temp;
        return;
    }
}
// 用'|'分割出多条命令，'|'最好用空格隔开
void parse(char *line, int *commandc, cmd **commands){
    int count = 0, i;
    char *p = NULL;
    if(line == NULL || strlen(line) == 0)
        return;

    char *pNext = line;

    while(1){
        my_strtok(&p, &pNext, '|');
        str2cmd(p, commands[count]);        // 会在原字符串添加一些'\0'来
分割
        //print_parse(commands[count]);
        ++count;
        if(pNext == NULL)
            break;
    }
    *commandc = count;
}
// 打印一条指令信息
void print_parse(cmd *com){
    printf("argc: %d\n", com->argc);
    int i;
    for(i = 0; i<com->argc; i++){
        printf("argv[%d]: %s\n", i, com->argv[i]);
    }
    printf("input: %s\n", com->input);
    printf("output: %s\n", com->output);
}
// 执行单独的一个管道命令
void pipe_sys(cmd *com){
    pid_t pid = fork();
    if(pid == 0){
        if(flag == 0){
```

```
            if(com->input != NULL)                  // 输入重定向
                if((fdin = open(com->input, O_RDONLY, 0666)) == -1){
                    puts("no such file or directory");
                    exit(1);
                }
                else{
                    dup2(fdin, 0);
                    close(fdin);
                }
            else
                dup2(recover[0], 0);
            dup2(fd[1], 1);                         // 输出重定向
            close(fd[0]);
            close(fd[1]);
            //puts("test_pipe");
            execvp(com->argv[0], com->argv);
            exit(0);
        }
        else if(flag == 1){
            dup2(fd[0], 0);                         // 输入重定向
            close(fd[0]);
            close(fd[1]);
            if(com->output != NULL){                // 输出重定向
                fdout = open(com->output, O_RDWR|O_CREAT|O_TRUNC, 0666);
                dup2(fdout, 1);
                close(fdout);
            }
            else{
                dup2(recover[1], 1);
            }
            //char str[10];
            //scanf("%s", str);
            //puts(str);
            execvp(com->argv[0], com->argv);
            exit(0);
        }
        else if(flag ==2){
            dup2(fd[0], 0);
            close(fd[0]);
            close(fd[1]);
            // 输出到临时管道
            dup2(fd_temp[1], 1);
            close(fd_temp[0]);
            close(fd_temp[1]);
```

```
            //char str[10];
            //scanf("%s", str);
            //puts(str);
            execvp(com->argv[0], com->argv);
            exit(0);
        }
    }
    wait(NULL);
    return;
}
// 处理并执行指令
void mysys(char *com){
    pid_t pid;
    int i, commandc, fd_chgr[2];
    cmd *commands[MAX_COMMAND];
    for(i = 0; i<MAX_COMMAND; i++)
        commands[i] = (cmd *)malloc(sizeof(cmd));
    char *c = (char *)malloc(sizeof(char)*strlen(com));
    strcpy(c, com);
    parse(c, &commandc, commands);
    //for(i = 0; i<commandc+1; i++){
    //  print_parse(commands[i]);
    //}

    pipe(fd);
    pipe(fd_temp);
    recover[0] = dup(0);
    recover[1] = dup(1);

    if(commandc == 1){      // 没有管道的情况
        if(pid = fork() == 0){
            if(commands[0]->input != NULL)          // 输入重定向
                if((fdin = open(commands[0]->input, O_RDONLY, 0666)) ==
-1){
                    puts("no such file or directory");
                    exit(1);
                }
                else{
                    dup2(fdin, 0);
                    close(fdin);
                }
            else
                dup2(recover[0], 0);
            if(commands[0]->output != NULL){        // 输出重定向
```

```
                fdout = open(commands[0]->output, O_RDWR|O_CREAT|O_TRUNC,
0666);
                dup2(fdout, 1);
                close(fdout);
            }
            else{
                dup2(recover[1], 1);
            }
            //print_parse(commands[0]);
            //run_cmd(commands[0]->argc, commands[0]->argv);
            execvp(commands[0]->argv[0], commands[0]->argv);
        }
        else{
            wait(NULL);
        }
    }
    else{           // 有管道的情
况       ////////////////////////////////////bug
        flag = -1;
        for(i = 0; i<commandc; i++){
            if(flag == 2){
                dup2(fd_temp[0], fd[0]);
                dup2(fd_temp[1], fd[1]);
                close(fd_temp[0]);
                close(fd_temp[1]);
                pipe(fd_temp);
                close(fd[1]);
            }
            if(flag == 0)
                close(fd[1]);
            if(i == 0)
                flag = 0;
            else if(i == commandc-1)
                flag = 1;
            else
                flag = 2;
            pipe_sys(commands[i]);
        }
    }
    exit(0);
}
// 没有实现识别输出重定向前的文件名
int main(){
    char dir[256];
```

```
    while(1){
        getcwd(dir, 254);
        strcat(dir, "\033[34;1m > \033[0m");
        if(command){
            free(command);
            command = NULL;
        }
        readline(dir);
        if(strstr(command, "cd ")){     // 注意 cd 后面加了一个空格
            if(strlen(command) > 3){
                sscanf(command, "cd %s", dir);
                chdir(dir);
            }
        }
        else if(strcmp(command, "pwd") == 0){
            getcwd(dir, 256);
            puts(dir);
        }
        else if(strcmp(command, "exit") == 0)
            exit(0);
        else{
            pipe(fd);
            pipe(fd_temp);
            recover[0] = dup(0);
            recover[1] = dup(1);
            int pid = fork();
            if(pid == 0){
                mysys(command);
            }
            else wait(NULL);
        }
    }
}
```

6.实验题目

pi1.c 使用 2 个线程根据莱布尼兹级数计算 π

实现思路

莱布尼兹级数公式: 1 - 1/3 + 1/5 - 1/7 + 1/9 - ... = PI/4

主线程创建一个辅助线程，辅助线程计算级数前半部分，主线程计算级数后半部分，最终将两部分结果相加输出。

运行结果

[karl@iZbncwqakj4ds9Z pi1]$ ./a.out

10000

3.1414926536

源代码

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

int n;      // n 表示数列求和的个数
// a_x = 2*x-1
void *compute(void *arg){
    int *a = (int *)arg;
    int i = 0;
    double *sum;
    *sum = 0;
    int k = 0;
    for(i = *a; i<n; i++){
        if((i+1)%2 == 0)
            k = (-1)*((i+1)*2-1);
        else
            k = ((i+1)*2-1);
        *sum += 1.0/(double)k;
    }
    return sum;
}
int main(){
    int mid = 0;
    scanf("%d", &n);
    mid = n/2;
    void *arg = &mid;
    pthread_t tid;
    pthread_create(&tid, NULL, compute, arg);

    int i = 0;
    double pi = 0;
    int k = 0;
    for(i = 0; i<mid; i++){
        if((i+1)%2 == 0)
            k = -1*((i+1)*2-1);
        else
            k = ((i+1)*2-1);
        pi += 1.0/(double)k;
    }

    double *r;
    pthread_join(tid, (void **)&r);
    printf("%.10lf\n", (pi+(*r))*4);
```

```
}
```

7.实验题目

Pi2.c 使用 N 个线程根据莱布尼兹级数计算 PI

实现思路

定义一个 param 参数，包含 start 和 end 参数，用 pthread_create 函数创建 N 个线程，为每个线程划分计算的区间并以参数传递。

运行结果

[karl@iZbncwqakj4ds9Z pi2]$ ./a.out

100000

3.1415826536

源代码

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

#define WORKERS 4
// a_x = 2*x-1
typedef struct param{
    int start;
    int end;
}param;
typedef struct result{
    double sum;
}result;
void *compute(void *arg){
    param *p = (param *)arg;
    int start = p->start;
    int end = p->end;
    double sum = 0;
    int i = 0;
    int k = 0;
    for(i = start; i<=end; i++){
        if(i%2 == 0)
            k = (-1)*(i*2-1);
        else
            k = (i*2-1);
        sum += 1.0/(double)k;
    }
    result *r = (result *)malloc(sizeof(result));
    r->sum = sum;
    return r;
}
```

```c
int main(){
    int i = 0;
    int n;      // n 表示数列求和的个数
    scanf("%d", &n);
    int start[WORKERS];
    int end[WORKERS];
    int average = n/WORKERS;
    int temp = 1;
    for(i = 0; i<WORKERS; i++){
        start[i] = temp;
        end[i] = start[i]+average;
        temp += (average+1);
    }
    end[WORKERS-1] = n;

    // 创建线程
    pthread_t tid[WORKERS];
    for(i = 0; i<WORKERS; i++){
        param *arg = (param *)malloc(sizeof(param));
        arg->start = start[i];
        arg->end = end[i];

        pthread_create(&tid[i], NULL, compute, arg);
    }

    // 求和
    double sum = 0;
    result *r;
    for(i = 0; i<WORKERS; i++){
        pthread_join(tid[i], (void **)&r);
        sum += ((result *)r)->sum;
        // printf("%f\n", ((result *)r)->sum);
    }

    printf("%.10lf\n", sum*4);
}
```

8.实验题目

sort.c 多线程排序

实现思路

主线程创建一个辅助线程。

主线程对数组前半部分排序。

辅助线程对数组后半部分排序。

使用归并排序合并两部分结果。

运行结果

```
[karl@iZbncwqakj4ds9Z sort]$ ./a.out
0   3   5   10  10  14  15  15  18  18  18  21  26  27  30  30  31  32  33
36  40  44  46  48  49  49  54  58  60  60  66  67  70  72  73  75  76  77
77  78  84  87  91  94  94  94  98  98  105 106 109 113 113 114 116 116 116
117 121 121 127 131 132 133 134 136 137 147 150 151 152 152 154 155 156
156 156 159 161 163 163 164 165 167 167 168 173 176 176 176 176 177 178
180 182 186 189 194 196 196
```

源代码

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

#define SIZE 100
int numbers[SIZE];
int result[SIZE];
void select_sort(int start, int end){
    int i, j;
    for(i = start; i<end; i++){
        int min = i;
        for(j = i; j<=end; j++)
            if(numbers[j] < numbers[min])
                min = j;
        int temp = numbers[i];
        numbers[i] = numbers[min];
        numbers[min] = temp;
    }
}
void *start_thread(void *arg){
    select_sort(SIZE/2+1, SIZE-1);

    return NULL;
}
int main(){
    srand(time(NULL));

    // 随机赋初值
    int i = 0;
    for(i = 0; i<SIZE; i++){
        numbers[i] = rand()%200;
    }

    // 创建线程
```

```
    pthread_t tid;
    pthread_create(&tid, NULL, start_thread, NULL);

    // 排序前半部分
    select_sort(0, SIZE/2);

    // 等待线程
    pthread_join(tid, NULL);

    // 归并排序 0-(SIZE/2) & (SIZE/2+1)-SIZE
    int p = 0, q = SIZE/2+1, r = 0;
    while(p <= SIZE/2 || q < SIZE){
        if(p > SIZE/2 && q < SIZE) result[r++] = numbers[q++];
        if(p <= SIZE/2 && q >= SIZE) result[r++] = numbers[p++];
        if(p <= SIZE/2 && q < SIZE)
            result[r++] = numbers[p] < numbers[q] ? numbers[p++] :
numbers[q++];
    }

    // 输出排序结果
    for(i = 0; i<SIZE; i++){
        printf("%d\t", result[i]);
    }
    puts("");

    return 0;
}
```

9.实验题目

Pc1.c 使用条件变量解决生产者、计算者、消费者问题

实现思路

生产者过程：获取 mutex1，阻塞在 empty1，若接收到 empty1 信号则生成一个产品，发出 full1 信号，释放 mutex1 锁。

计算者过程：获取 mutex1，阻塞在 full1，若接收到 full1 信号则从队列一取出一个产品，发出 empty1 信号，释放 mutex1 锁。然后获取 mutex2 锁，阻塞在 empty2 信号，若接收到 empty2 信号则将计算后的产品放到队列二，发出 full2 信号，释放 mutex2 锁。

消费者过程：获取 mutex2，阻塞在 full2，若接收到 full2 信号则从队列 2 取出一个产品，发出 empty2 信号，释放 mutex2 锁。

运行结果

[karl@iZbncwqakj4ds9Z pc1]$ ./a.out
produce a
produce b
produce c
compute A

compute B
compute C
consume A
consume B
consume C
produce d
produce e
produce f
compute D
compute E
compute F
consume D
consume E
consume F
produce g
produce h
compute G
compute H
consume G
consume H

源代码

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

#define CAPACITY 4
#define ITEM_COUNT 8
int buffer1[CAPACITY];
int buffer2[CAPACITY];
int in1, in2;
int out1, 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;
}
void buffer1_put(int item){
    buffer1[in1] = item;
```

```
    in1 = (in1 + 1)%CAPACITY;
}
void buffer2_put(int item){
    buffer2[in2] = item;
    in2 = (in2 + 1)%CAPACITY;
}
int buffer1_get(){
    int item;
    item = buffer1[out1];
    out1 = (out1 + 1)%CAPACITY;
    return item;
}
int buffer2_get(){
    int item;
    item = buffer2[out2];
    out2 = (out2 + 1)%CAPACITY;
    return item;
}
void print_buffer(){
    int i;
    for(i = 0; i<CAPACITY; i++){
        printf("%c ", buffer1[i]);
    }
    puts("\n1");
    for(i = 0; i<CAPACITY; i++){
        printf("%c ", buffer2[i]);
    }
    puts("\n2");
}
pthread_cond_t empty1, empty2;
pthread_cond_t full1, full2;
pthread_mutex_t mutex1, mutex2;
void *producer(void *arg){
    int i;
    for(i = 0; i<ITEM_COUNT; i++){
        pthread_mutex_lock(&mutex1);
        while(buffer1_is_full()){
            pthread_cond_wait(&empty1, &mutex1);
        }
        buffer1_put('a'+i);
        printf("produce %c\n", 'a'+i);

        pthread_cond_signal(&full1);
        pthread_mutex_unlock(&mutex1);
```

```
    }
    return;
}
void *computer(void *arg){
    int i, item;
    for(i = 0; i<ITEM_COUNT; i++){
        pthread_mutex_lock(&mutex1);
        while(buffer1_is_empty()){
            pthread_cond_wait(&full1, &mutex1);
        }
        item = buffer1_get();
        pthread_cond_signal(&empty1);
        pthread_mutex_unlock(&mutex1);

        item += 'A'-'a';

        pthread_mutex_lock(&mutex2);
        while(buffer2_is_full()){
            pthread_cond_wait(&empty2, &mutex2);
        }
        buffer2_put(item);
        printf("compute %c\n", item);
        pthread_cond_signal(&full2);
        pthread_mutex_unlock(&mutex2);

        //print_buffer();
    }
    return;
}
void *consumer(void *arg){
    int i;
    int item;
    for(i = 0; i<ITEM_COUNT; i++){
        pthread_mutex_lock(&mutex2);
        while(buffer2_is_empty()){
            pthread_cond_wait(&full2, &mutex2);
        }
        item = buffer2_get();
        printf("consume %c\n", item);

        pthread_cond_signal(&empty2);
        pthread_mutex_unlock(&mutex2);
    }
    return;
```

```
}
int main(){
    pthread_t prd_tid, cpt_tid, csm_tid;
    pthread_cond_init(&empty1, NULL);
    pthread_cond_init(&empty2, NULL);
    pthread_cond_init(&full1, NULL);
    pthread_cond_init(&full2, NULL);
    pthread_mutex_init(&mutex1, NULL);
    pthread_mutex_init(&mutex2, NULL);

    pthread_create(&prd_tid, NULL, producer, NULL);
    pthread_create(&cpt_tid, NULL, computer, NULL);
    pthread_create(&csm_tid, NULL, consumer, NULL);

    pthread_join(prd_tid, NULL);
    pthread_join(cpt_tid, NULL);
    pthread_join(csm_tid, NULL);

    return 0;
}
```

10.实验题目

Pc2.c 使用信号量解决生产者、计算者、消费者问题

实现思路

生产者等待 empty1 信号量，等待 mutex1 信号量，向队列一生成一个产品，发出 mutex1 信号量，发出 full1 信号量。

计算者等待 full1 信号量，等待 mutex1 信号量，从队列一取出一个产品，发出 mutex1 信号量，发出 empty1 信号量。计算。然后等待 empty2 信号量，等待 mutex2 信号量，向队列二放入计算结果，发出 mutex2 信号量，发出 full2 信号量。

消费者等待 full2 信号量，等待 mutex2 信号量，从队列二取出一个产品，发出 mutex2 信号量，发出 empty2 信号量。

运行结果

```
[karl@iZbncwqakj4ds9Z pc2]$ ./a.out
produce a
produce b
produce c
compute A
compute B
compute C
consume A
consume B
consume C
produce d
produce e
```

produce f
compute D
compute E
compute F
consume D
consume E
consume F
produce g
produce h
compute G
compute H
consume G
consume H

源代码

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

#define CAPACITY 4
int buffer1[CAPACITY], buffer2[CAPACITY];
int in1, in2;
int out1, 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;
}
void buffer1_put(int item){
    buffer1[in1] = item;
    in1 = (in1 + 1)%CAPACITY;
}
void buffer2_put(int item){
    buffer2[in2] = item;
    in2 = (in2 + 1)%CAPACITY;
}
int buffer1_get(){
    int item;
    item = buffer1[out1];
```

```
    out1 = (out1 + 1)%CAPACITY;
    return item;
}
int buffer2_get(){
    int item;
    item = buffer2[out2];
    out2 = (out2 + 1)%CAPACITY;
    return item;
}
void print_buffer(){
    int i;
    for(i = 0; i<CAPACITY; i++){
        printf("%c ", buffer1[i]);
    }
    puts("\n1");
    for(i = 0; i<CAPACITY; i++){
        printf("%c ", buffer2[i]);
    }
    puts("\n2");
}
typedef struct sema_t{
    int value;
    pthread_mutex_t mutex;
    pthread_cond_t cond;
}sema_t;
void sema_init(sema_t *sema, int value){
    sema->value = value;
    pthread_mutex_init(&sema->mutex, NULL);
    pthread_cond_init(&sema->cond, NULL);
}
void sema_wait(sema_t *sema){
    pthread_mutex_lock(&sema->mutex);
    while(sema->value <= 0){
        pthread_cond_wait(&sema->cond, &sema->mutex);
    }
    sema->value--;
    pthread_mutex_unlock(&sema->mutex);
}
void sema_signal(sema_t *sema){
    pthread_mutex_lock(&sema->mutex);
    ++sema->value;
    pthread_cond_signal(&sema->cond);
    pthread_mutex_unlock(&sema->mutex);
}
```

```
sema_t mutex1, mutex2;
sema_t empty1, empty2;
sema_t full1, full2;
#define ITEM_COUNT 8
void *producer(){
    int i;
    for(i = 0; i<ITEM_COUNT; i++){
        sema_wait(&empty1);
        sema_wait(&mutex1);
        buffer1_put('a'+i);
        printf("produce %c\n", 'a'+i);
        sema_signal(&mutex1);
        sema_signal(&full1);
    }
}
void *computer(){
    int i, item;
    for(i = 0; i<ITEM_COUNT; i++){
        sema_wait(&full1);
        sema_wait(&mutex1);
        item = buffer1_get();
        sema_signal(&mutex1);
        sema_signal(&empty1);

        item += 'A'-'a';

        sema_wait(&empty2);
        sema_wait(&mutex2);
        buffer2_put(item);
        printf("compute %c\n", item);
        sema_signal(&mutex2);
        sema_signal(&full2);
    }
}
void *consumer(){
    int i, item;
    for(i = 0; i<ITEM_COUNT; i++){
        sema_wait(&full2);
        sema_wait(&mutex2);
        item = buffer2_get();
        printf("consume %c\n", item);
        sema_signal(&mutex2);
        sema_signal(&empty2);
    }
```

```
}
int main(){
    pthread_t prd_tid, cpt_tid, csm_tid;

    sema_init(&mutex1, 1);
    sema_init(&mutex2, 1);
    sema_init(&empty1, CAPACITY-1);
    sema_init(&empty2, CAPACITY-1);
    sema_init(&full1, 0);
    sema_init(&full2, 0);

    pthread_create(&prd_tid, NULL, producer, NULL);
    pthread_create(&cpt_tid, NULL, computer, NULL);
    pthread_create(&csm_tid, NULL, consumer, NULL);

    pthread_join(prd_tid, NULL);
    pthread_join(cpt_tid, NULL);
    pthread_join(csm_tid, NULL);

    return 0;
}
```

11.实验题目

ring.c 创建 N 个线程，它们构成一个环

实现思路

第一个线程：获取后一个线程的锁，向第二个数组元素填 1，释放后一个线程的锁，释放后一线程的信号量。然后等待自己的信号量和锁，获取第一个数组元素的值，释放自己的锁。

中间部分的线程：获取此线程的信号量和锁，接受自己的数组元素，释放自己的锁。然后获取后一个线程的锁，向后一个数组元素填写加 1 后的结果，释放后一个线程的锁和信号量。

最后一个线程：获取此线程的信号量和锁，接受自己的数组元素，释放自己的锁。然后获取第一个线程的锁，向第一个数组元素填写加 1 后的结果，释放第一个线程的锁和信号量。

运行结果

```
[karl@iZbncwqakj4ds9Z ring]$ ./a.out
T1 send 1
T2 received 1
T2 send 2
T3 received 2
T3 send 3
T4 received 3
T4 send 4
T1 received 4
```

源代码

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

#define N 4
int buffer[N];
typedef struct sema_t{
    int value;
    pthread_mutex_t mutex;
    pthread_cond_t cond;
}sema_t;
typedef struct param{
    int order;
}param;
void sema_init(sema_t *sema, int value){
    sema->value = value;
    pthread_mutex_init(&sema->mutex, NULL);
    pthread_cond_init(&sema->cond, NULL);
}
void sema_wait(sema_t *sema){
    pthread_mutex_lock(&sema->mutex);
    while(sema->value <= 0){
        pthread_cond_wait(&sema->cond, &sema->mutex);
    }
    sema->value--;
    pthread_mutex_unlock(&sema->mutex);
}
void sema_signal(sema_t *sema){
    pthread_mutex_lock(&sema->mutex);
    ++sema->value;
    pthread_cond_signal(&sema->cond);
    pthread_mutex_unlock(&sema->mutex);
}
sema_t mutex[N];
sema_t full[N];
void *porter(void *arg){
    int receive;
    param *p = (param *)arg;
    int order = p->order;
    if(order == 0){          // 第一个节点
        sema_wait(&mutex[order+1]);
        buffer[order+1] = 1;
        printf("T1 send 1\n");
        sema_signal(&mutex[order+1]);
```

```
        sema_signal(&full[order+1]);

        sema_wait(&full[0]);
        sema_wait(&mutex[0]);
        receive = buffer[0];
        printf("T%d received %d\n", 1, receive);
        sema_signal(&mutex[0]);
        exit(0);
    }
    else if(order == N-1){  // 最后一个节点
        sema_wait(&full[order]);
        sema_wait(&mutex[order]);
        receive = buffer[order];
        printf("T%d received %d\n", order+1, receive);
        sema_signal(&mutex[order]);

        sema_wait(&mutex[0]);
        buffer[0] = receive+1;
        printf("T%d send %d\n", order+1, receive+1);
        sema_signal(&mutex[0]);
        sema_signal(&full[0]);
    }
    else{                   // 中间节点
        sema_wait(&full[order]);
        sema_wait(&mutex[order]);
        receive = buffer[order];
        printf("T%d received %d\n", order+1, receive);
        sema_signal(&mutex[order]);

        sema_wait(&mutex[order+1]);
        buffer[order+1] = receive+1;
        printf("T%d send %d\n", order+1, receive+1);
        sema_signal(&mutex[order+1]);
        sema_signal(&full[order+1]);
    }
}
int main(){
    pthread_t porter_tid[N];
    param p[N];
    int i;
    for(i = 0; i<N; i++){
        sema_init(&mutex[i], 1);
        sema_init(&full[i], 0);
    }
```

```
    for(i = 0; i<N; i++){
        p[i].order = i;
        pthread_create(&porter_tid[i], NULL, porter, &p[i]);
    }
    for(i = 0; i<N; i++){
        pthread_join(porter_tid[i], NULL);
    }
    return 0;
}
```