姓名：吴涵　　学号：202101000720　　成绩：

实验 28 父子进程线程异步性

1、创建父子进程，观察父子进程执行的顺序，了解进程执行的异步行为

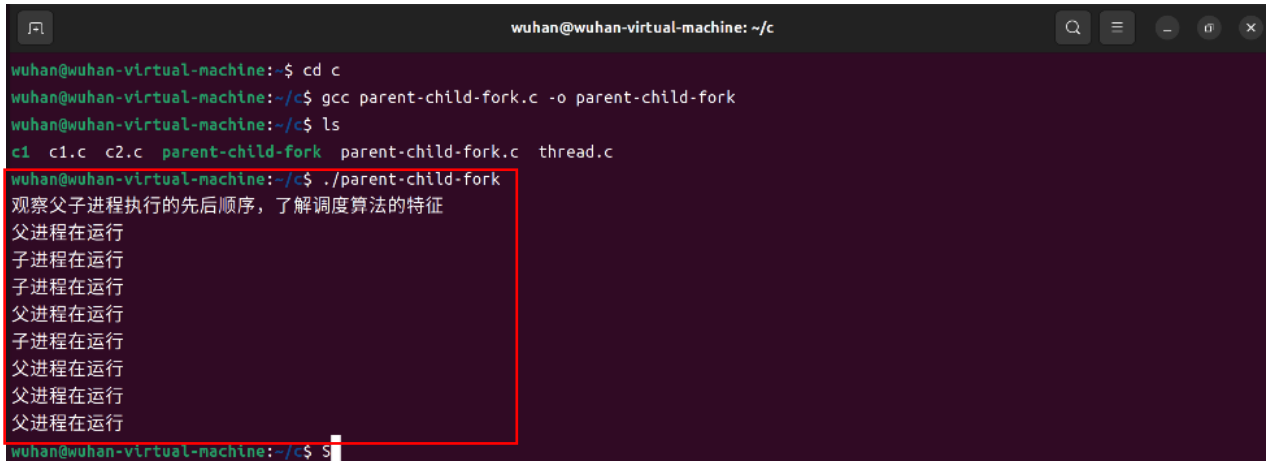
源程序：

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
    pid_t pid;
    char*msg;
    int k;
    printf("观察父子进程执行的先后顺序，了解调度算法的特征\n");
    pid=fork();
    switch(pid)
    {
        case 0:
            msg="子进程在运行";
            k=3;
            break;
        case -1:
            msg="进程创建失败";
            break;
        default:
            msg="父进程在运行";
            k=5;
            break;
    }
    while(k>0)
    {
        puts(msg);
        sleep(1);
        k--;
     }
     exit(0);
}
```

编译链接命令：gcc parent-child-fork.c -o parent-child-fork

运行命令：./parent-child-fork

交互与结果：



2、创建主线程和子线程，观察多线程执行的顺序，了解线程执行的异步行为

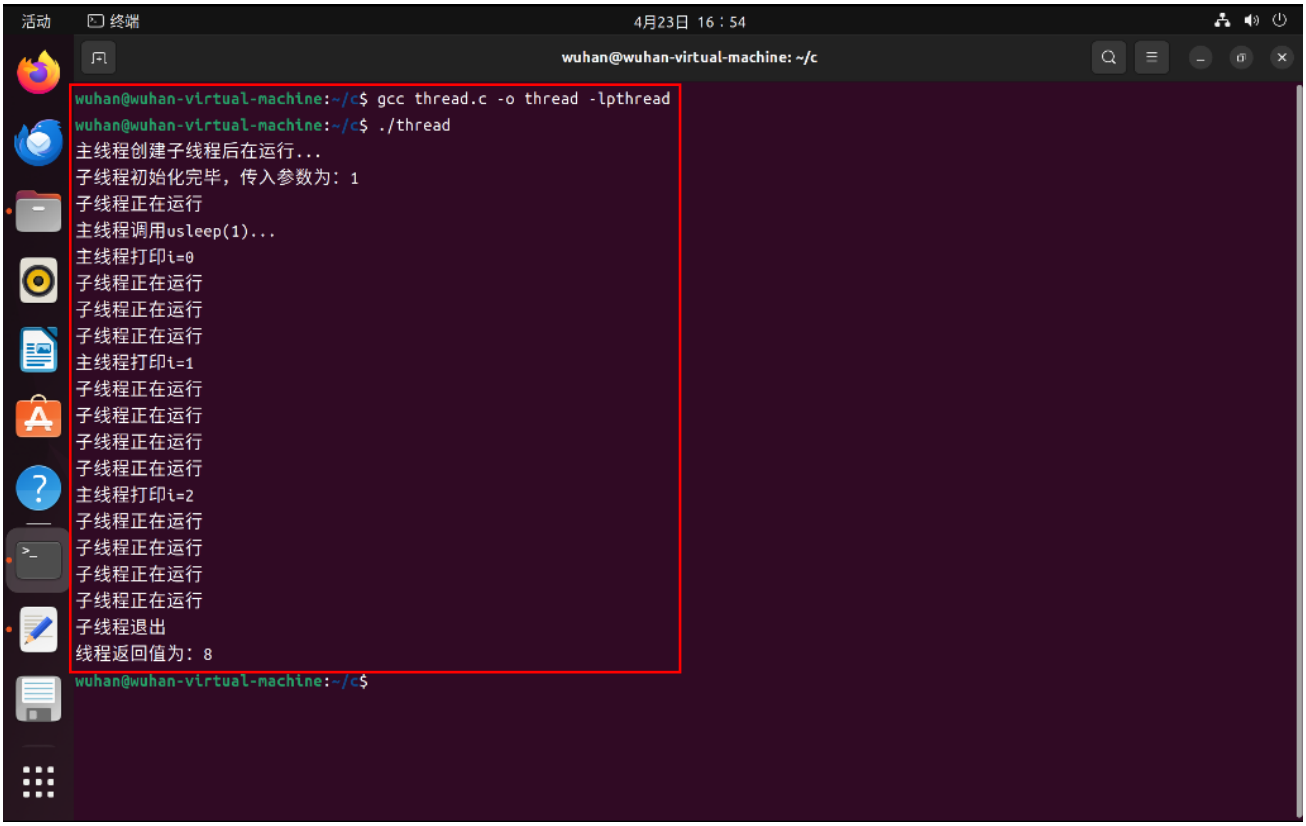
源程序：

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
static int run=1;
static int retvalue;
void *threadfunc(void*arg)
{
        int*running=arg;
        printf("子线程初始化完毕，传入参数为：%d\n",*running);
        while(*running)
        {
                printf("子线程正在运行\n");
                usleep(1);
        }
        printf("子线程退出\n");
        retvalue=8;
        pthread_exit((void*)&retvalue);
}
int main()
{
        pthread_t pt;
        int ret=-1;
        int times=3;
        int i=0;
        int *ret_join=NULL;
        ret=pthread_create(&pt,NULL,(void*)threadfunc,&run);
        if(ret!=0)
        {
                printf("建立线程失败\n");
                return 1;
        }
        printf("主线程创建子线程后在运行...\n");
        usleep(1);
        printf("主线程调用 usleep(1)...\n");
        for(;i<times;i++)
        {
                printf("主线程打印 i=%d\n",i);
                usleep(1);
        }
        run=0;
        pthread_join(pt,(void*)&ret_join);
        printf("线程返回值为：%d\n",*ret_join);
        return 0;
}
```

编译链接命令：gcc thread.c -o thread -lpthread

运行命令：./thread

交互与结果：


活动
终端
4月23日 16：54
wuhan@wuhan-virtual-machine: ~/c
wuhan@wuhan-virtual-machine:~/c$ gcc thread.c -o thread -lpthread
wuhan@wuhan-virtual-machine:~/c$ ./thread
主线程创建子线程后在运行...
子线程初始化完毕，传入参数为：1
子线程正在运行
主线程调用usleep(1)...
主线程打印i=0
子线程正在运行
子线程正在运行
子线程正在运行
主线程打印i=1
子线程正在运行
子线程正在运行
子线程正在运行
子线程正在运行
主线程打印i=2
子线程正在运行
子线程正在运行
子线程正在运行
子线程正在运行
子线程退出
线程返回值为：8
wuhan@wuhan-virtual-machine:~/c$

3、多线程对共享变量的非互斥访问

源程序：

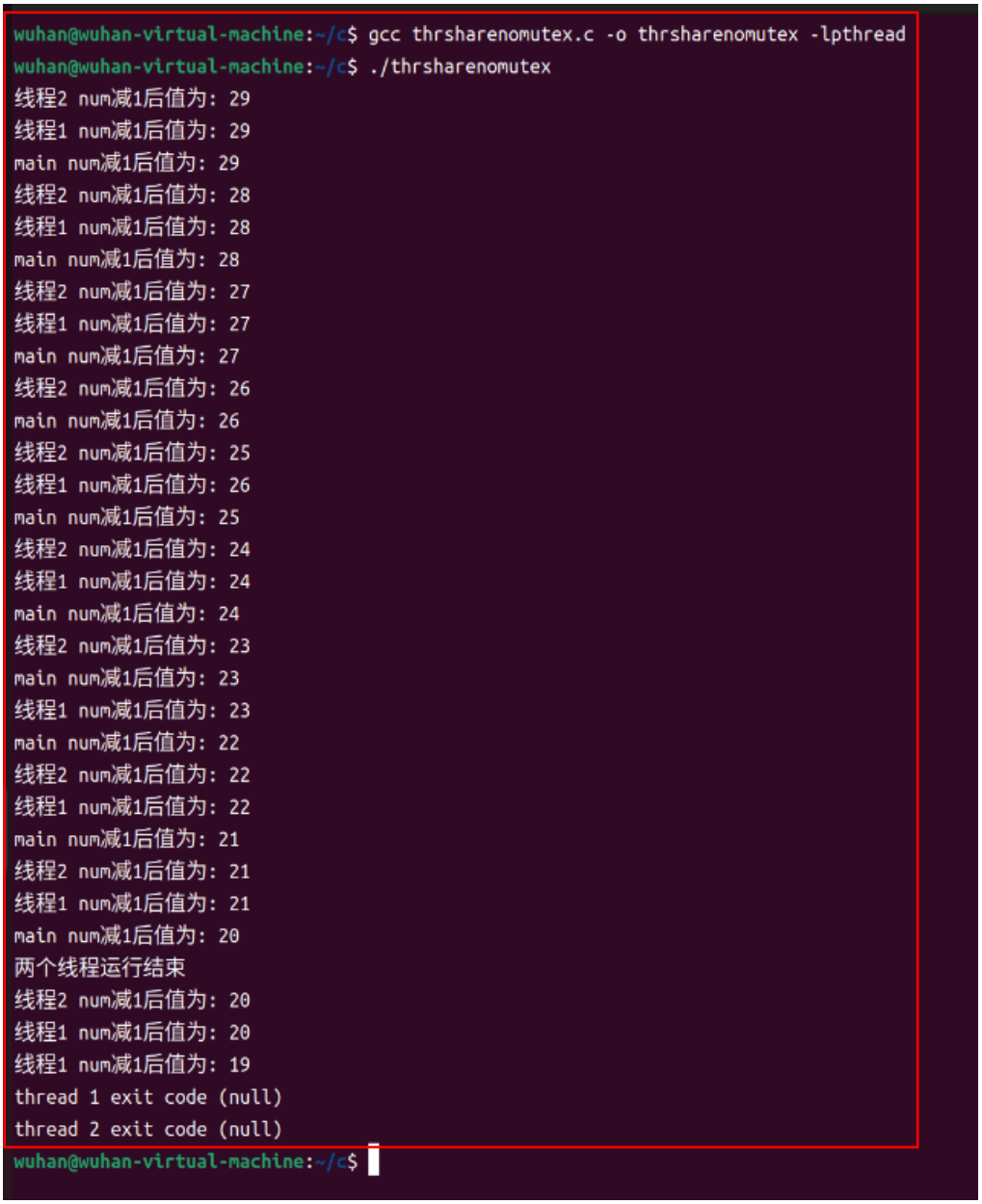
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <string.h>
int num = 30, count = 10;
void *sub1(void *arg)
{
    int i = 0, tmp;
    for (; i < count; i++)
    {
        tmp = num - 1;
        usleep(13);
        num = tmp;
        printf("线程 1 num 减 1 后值为: %d\n", num);
    }
    return ((void *)0);
}
void *sub2(void *arg)
{
    int i = 0, tmp;
    for (; i < count; i++)
    {
        tmp = num - 1;
        usleep(31);
        num = tmp;
        printf("线程 2 num 减 1 后值为: %d\n", num);
    }
    return ((void *)0);
}
int main(int argc, char **argv)
{
    pthread_t tid1, tid2;
    int err, i = 0, tmp;
    void *tret;
    err = pthread_create(&tid1, NULL, sub1, NULL);
    if (err != 0)
    {
        printf("pthread_create error:%s\n", strerror(err));
        exit(-1);
    }
    err = pthread_create(&tid2, NULL, sub2, NULL);
    if (err != 0)
    {
        printf("pthread_create error:%s\n", strerror(err));
        exit(-1);
    }
    for (; i < count; i++)
    {
        tmp = num - 1;
        usleep(5);
        num = tmp;
        printf("main num 减 1 后值为: %d\n", num);
    }
    printf("两个线程运行结束\n");
```

```
    err = pthread_join(tid1, &tret);
    if (err != 0)
    {
        printf("can not join with thread1:%s\n", strerror(err));
        exit(-1);
    }
    printf("thread 1 exit code %d\n", (int)tret);
    err = pthread_join(tid2, &tret);
    if (err != 0)
    {
        printf("can not join with thread1:%s\n", strerror(err));
        exit(-1);
    }
    printf("thread 2 exit code %d\n", (int)tret);
    return 0;
}
```

编译链接命令：gcc thrsharenomutex.c -o thrsharenomutex -lpthread

运行命令：./thrsharenomutex

交互与结果：



实验 29 同步与互斥

1、并发线程同步与互斥

源程序：

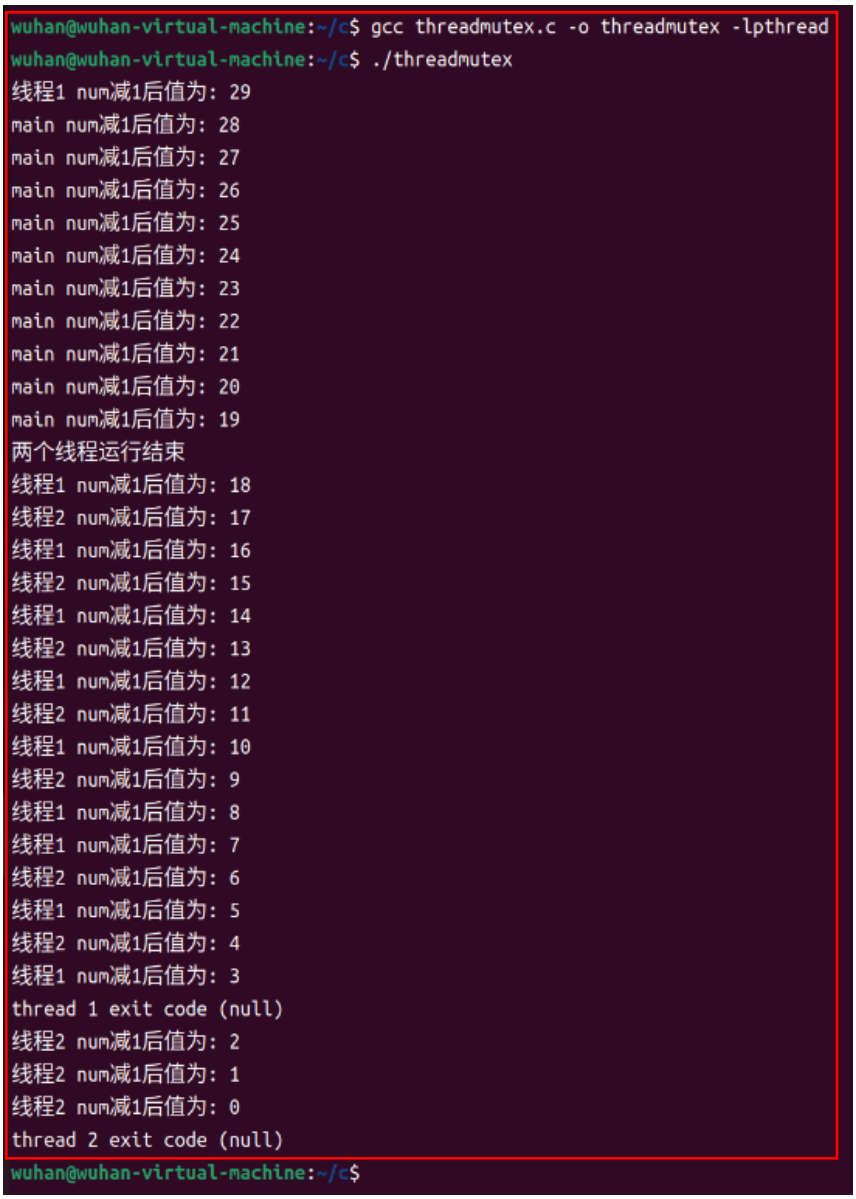
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <string.h>
int num = 30, count = 10;
pthread_mutex_t mylock = PTHREAD_MUTEX_INITIALIZER;
void *sub1(void *arg)
{
    int i = 0, tmp;
    for (; i < count; i++)
    {
        pthread_mutex_lock(&mylock);
        tmp = num - 1;
        usleep(13);
        num = tmp;
        pthread_mutex_unlock(&mylock);
        printf("线程 1 num 减 1 后值为: %d\n", num);
    }
    return ((void *)0);
}
void *sub2(void *arg)
{
    int i = 0, tmp;
    for (; i < count; i++)
    {
        pthread_mutex_lock(&mylock);
        tmp = num - 1;
        usleep(31);
        num = tmp;
        pthread_mutex_unlock(&mylock);
        printf("线程 2 num 减 1 后值为: %d\n", num);
    }
    return ((void *)0);
}
int main(int argc, char **argv)
{
    pthread_t tid1, tid2;
    int err, i = 0, tmp;
    void *tret;
    err = pthread_create(&tid1, NULL, sub1, NULL);
    if (err != 0)
    {
        printf("pthread_create error:%s\n", strerror(err));
        exit(-1);
    }
    err = pthread_create(&tid2, NULL, sub2, NULL);
    if (err != 0)
    {
        printf("pthread_create error:%s\n", strerror(err));
        exit(-1);
    }
    for (; i < count; i++)
    {
        pthread_mutex_lock(&mylock);
        tmp = num - 1;
        usleep(5);
        num = tmp;
        pthread_mutex_unlock(&mylock);
        printf("main num 减 1 后值为: %d\n", num);
```

```
    }
    printf("两个线程运行结束\n");
    err = pthread_join(tid1, &tret);
    if (err != 0)
    {
        printf("can not join with thread1:%s\n", strerror(err));
        exit(-1);
    }
    printf("thread 1 exit code %d\n", (int)tret);
    err = pthread_join(tid2, &tret);
    if (err != 0)
    {
        printf("can not join with thread1:%s\n", strerror(err));
        exit(-1);
    }
    printf("thread 2 exit code %d\n", (int)tret);
    return 0;
}
```

编译链接命令：gcc threadmutex.c -o threadmutex -lpthread

运行命令：./threadmutex

交互与结果：



2、生产者-消费者同步与互斥试验

源程序：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <pthread.h>
#include <unistd.h>
#include <signal.h>
#include <semaphore.h>
#define Maxbuf 10    // 缓冲单元数目
#define TimesOfOp 10 // 生产者、消费者循环读写缓冲区的次数
#define true 1
#define false 0
#define historynum 100 // 生产者、消费者读写历史记录数目
struct Circlebuf       // 循环缓冲队列结构
{
    int read;        // 读指针
    int write;       // 写指针
    int buf[Maxbuf]; // 缓冲区
} circlebuf;
sem_t mutex;                                                    // 互斥信号量
sem_t empty;                                                    // 空白缓冲区同步信号
量
sem_t full;                                                     // 满缓冲区同步信号量
char writehistory[historynum][30];                              // 写历史
char readhistory[historynum][30];                               // 读历史
int writehistorycount = 0;                                      // 写历史计数器
int readhistorycount = 0;                                       // 读历史计数器
char history[historynum][30];                                   // 缓冲区操作历史
int historycount = 0;                                           // 缓冲区操作历史计数
器
void writeCirclebuf(struct Circlebuf *circlebuf, int *value) // 向缓冲区中写一个值
{
    circlebuf->buf[circlebuf->write] = (*value);
    sleep(1);
    circlebuf->write = (circlebuf->write + 1) % Maxbuf;
}
int readCirclebuf(struct Circlebuf *circlebuf)
{
    int value = 0;
    value = circlebuf->buf[circlebuf->read];
    sleep(1);
    circlebuf->buf[circlebuf->read] = 0;
    circlebuf->read = (circlebuf->read + 1) % Maxbuf;
    return value;
}
void sigend(int sig)
{
    exit(0);
}
void *productThread(void *i)
{
    int *n = (int *)i;
    int t = TimesOfOp;
    int writeptr;
    while (t--)
    {
```

```
        sem_wait(&empty);
        sem_wait(&mutex);
        writeCirclebuf(&circlebuf, n);
        if (circlebuf.write > 0)
            writeptr = circlebuf.write - 1;
        else
            writeptr = Maxbuf - 1;
        sprintf(writehistory[writehistorycount++], "生产者%d:缓冲区%d=%d", *n,
writeptr, *n);
        sprintf(history[historycount++], "生产者%d:缓冲区%d=%d\n", *n, writeptr,
*n);
        sem_post(&mutex);
        sem_post(&full);
        sleep(1);
    }
}
void *consumerThread(void *i)
{
    int *n = (int *)i;
    int t = TimesOfOp;
    int value = 0;
    int readptr;
    while (t--)
    {
        sem_wait(&full);
        sem_wait(&mutex);
        value = readCirclebuf(&circlebuf);
        if (circlebuf.read > 0)
            readptr = circlebuf.read - 1;
        else
            readptr = Maxbuf - 1;
        sprintf(readhistory[readhistorycount++], "消费者%d:缓冲区%d=%d\n", *n,
readptr, value);
        sprintf(history[historycount++], "消费者%d:缓冲区%d=%d\n", *n, readptr,
value);
        sem_post(&mutex);
        sem_post(&empty);
        sleep(1);
    }
}
int main()
{
    int i, max;
    int ConsNum = 0, ProdNum = 0, ret;
    sem_init(&mutex, 0, 1);
    sem_init(&empty, 0, Maxbuf);
    sem_init(&full, 0, 0);
    signal(SIGINT, sigend);
    signal(SIGTERM, sigend);
    circlebuf.read = circlebuf.write = 0;
    for (i = 0; i < Maxbuf; i++)
        circlebuf.buf[i] = 0;
    printf("请输入生产者线程的数目 :");
    scanf("%d", &ProdNum);
    int *pro = (int *)malloc(ProdNum * sizeof(int));
    pthread_t *proid = (pthread_t *)malloc(ProdNum * sizeof(pthread_t));
    printf("请输入消费者线程的数目 :");
    scanf("%d", &ConsNum);
```

```
    int *con = (int *)malloc(ConsNum * sizeof(int));
    pthread_t *conid = (pthread_t *)malloc(ConsNum * sizeof(pthread_t));
    for (i = 1; i <= ConsNum; i++)
    {
        con[i - 1] = i;
        ret = pthread_create(&conid[i], NULL, consumerThread, (void *)&con[i -
1]);
        if (ret != 0)
        {
            printf("Create thread error");
            exit(1);
        }
    }
    for (i = 1; i <= ProdNum; i++)
    {
        pro[i - 1] = i;
        ret = pthread_create(&proid[i], NULL, productThread, (void *)&pro[i -
1]);
        if (ret != 0)
        {
            printf("Create thread error");
            exit(1);
        }
    }
    sleep((ConsNum + ProdNum) * 10);
    if (writehistorycount > readhistorycount)
        max = writehistorycount;
    else
        max = readhistorycount;
    for (i = 0; i < max; i++)
        if ((i < writehistorycount) && (i < readhistorycount))
            printf("%s | %s\n", writehistory[i], readhistory[i]);
        else if (i < writehistorycount)
            printf("%s | %s\n", writehistory[i], " ");
        else
            printf("%s | %s\n", " ", readhistory[i]);
    printf("**************缓冲池的操作历史为：******************\n");
    for (i = 0; i < historycount; i++)
        printf("%s", history[i]);
    sem_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);
}
```

编译链接命令：gcc pc1.c -o pc1 -lpthread

运行命令：./pc1

交互与结果：（非全部截图）

```
^Cwuhan@wuhan-virtual-machine:~/c$ ./pc
请输入生产者线程的数目 :4
请输入消费者线程的数目 :3
生产者1:缓冲区0=1 | 消费者3:缓冲区0=1

生产者2:缓冲区1=2 | 消费者1:缓冲区1=2

生产者3:缓冲区2=3 | 消费者2:缓冲区2=3

生产者1:缓冲区3=1 | 消费者1:缓冲区3=1

生产者2:缓冲区4=2 | 消费者3:缓冲区4=2

生产者3:缓冲区5=3 | 消费者2:缓冲区5=3

生产者3:缓冲区6=3 | 消费者1:缓冲区6=3

生产者2:缓冲区7=2 | 消费者3:缓冲区7=2

生产者2:缓冲区8=2 | 消费者1:缓冲区8=2

生产者3:缓冲区9=3 | 消费者1:缓冲区9=3

生产者1:缓冲区0=1 | 消费者2:缓冲区0=1

生产者2:缓冲区1=2 | 消费者1:缓冲区1=2

生产者3:缓冲区2=3 | 消费者3:缓冲区2=3

生产者1:缓冲区3=1 | 消费者3:缓冲区3=1

生产者2:缓冲区4=2 | 消费者2:缓冲区4=2

生产者3:缓冲区5=3 | 消费者1:缓冲区5=3

生产者3:缓冲区6=3 | 消费者3:缓冲区6=3

生产者2:缓冲区7=2 | 消费者1:缓冲区7=2

生产者1:缓冲区8=1 | 消费者2:缓冲区8=1
```

```
生产者2:缓冲区6=2 | 消费者2:缓冲区6=2

生产者1:缓冲区7=1 | 消费者2:缓冲区7=1

生产者1:缓冲区8=1 | 消费者3:缓冲区8=1

生产者1:缓冲区9=1 | 消费者3:缓冲区9=1

生产者4:缓冲区0=4 |
生产者4:缓冲区1=4 |
生产者4:缓冲区2=4 |
生产者4:缓冲区3=4 |
生产者4:缓冲区4=4 |
************缓冲池的操作历史为：*****************
生产者1:缓冲区0=1
生产者2:缓冲区1=2
生产者3:缓冲区2=3
消费者3:缓冲区0=1
消费者1:缓冲区1=2
消费者2:缓冲区2=3
生产者1:缓冲区3=1
生产者2:缓冲区4=2
消费者1:缓冲区3=1
生产者3:缓冲区5=3
消费者3:缓冲区4=2
生产者3:缓冲区6=3
消费者2:缓冲区5=3
生产者2:缓冲区7=2
消费者1:缓冲区6=3
生产者2:缓冲区8=2
消费者3:缓冲区7=2
生产者3:缓冲区9=3
消费者1:缓冲区8=2
生产者1:缓冲区0=1
消费者1:缓冲区9=3
生产者2:缓冲区1=2
消费者2:缓冲区0=1
生产者3:缓冲区2=3
消费者1:缓冲区1=2
生产者1:缓冲区3=1
```

3、生产者-消费者未加同步与互斥机制的运行试验

源程序：

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

#define Maxbuf 10 // 缓冲单元数目
#define TimesOfOp 10 // 生产者、消费者循环读写缓冲区的次数
#define historynum 100 // 生产者、消费者读写历史记录数目

struct Circlebuf // 循环缓冲队列结构
{
    int read;        // 读指针
    int write;       // 写指针
    int buf[Maxbuf]; // 缓冲区
} circlebuf;

char writehistory[historynum][30]; // 写历史
char readhistory[historynum][30];  // 读历史
int writehistorycount = 0;         // 写历史计数器
int readhistorycount = 0;          // 读历史计数器
char history[historynum][30];      // 缓冲区操作历史
int historycount = 0;              // 缓冲区操作历史计数器

void writeCirclebuf(struct Circlebuf *circlebuf, int *value) // 向缓冲区中写一个值
{
    circlebuf->buf[circlebuf->write] = (*value);
    sleep(1);
    circlebuf->write = (circlebuf->write + 1) % Maxbuf;
}

int readCirclebuf(struct Circlebuf *circlebuf)
{
    int value = 0;
    value = circlebuf->buf[circlebuf->read];
    sleep(1);
    circlebuf->buf[circlebuf->read] = 0;
    circlebuf->read = (circlebuf->read + 1) % Maxbuf;
    return value;
}

void *productThread(void *arg)
{
    int *n = (int *)arg;
    int t = TimesOfOp;
    int writeptr;
    while (t--)
    {
        writeCirclebuf(&circlebuf, n);
        if (circlebuf.write > 0)
            writeptr = circlebuf.write - 1;
        else
            writeptr = Maxbuf - 1;
        sprintf(writehistory[writehistorycount++], "生产者%d:缓冲区%d=%d", *n,
writeptr, *n);
```

```
        sprintf(history[historycount++], "生产者%d:缓冲区%d=%d\n", *n, writeptr,
*n);
        sleep(1);
    }
}

void *consumerThread(void *arg)
{
    int *n = (int *)arg;
    int t = TimesOfOp;
    int value = 0;
    int readptr;
    while (t--)
    {
        value = readCirclebuf(&circlebuf);
        if (circlebuf.read > 0)
            readptr = circlebuf.read - 1;
        else
            readptr = Maxbuf - 1;
        sprintf(readhistory[readhistorycount++], "消费者%d:缓冲区%d=%d\n", *n,
readptr, value);
        sprintf(history[historycount++], "消费者%d:缓冲区%d=%d\n", *n, readptr,
value);
        sleep(1);
    }
}

int main()
{
    int i, max;
    int ConsNum = 0, ProdNum = 0, ret;

    circlebuf.read = circlebuf.write = 0;
    for (i = 0; i < Maxbuf; i++)
        circlebuf.buf[i] = 0;

    printf("请输入生产者线程的数目 :");
    scanf("%d", &ProdNum);

    pthread_t *proid = (pthread_t *)malloc(ProdNum * sizeof(pthread_t));
    int *proArgs = (int *)malloc(ProdNum * sizeof(int));
    for (i = 0; i < ProdNum; i++)
        proArgs[i] = i + 1;

    printf("请输入消费者线程的数目 :");
    scanf("%d", &ConsNum);

    pthread_t *conid = (pthread_t *)malloc(ConsNum * sizeof(pthread_t));
    int *conArgs = (int *)malloc(ConsNum * sizeof(int));
    for (i = 0; i < ConsNum; i++)
        conArgs[i] = i + 1;

    for (i = 0; i < ConsNum; i++)
    {
        ret = pthread_create(&conid[i], NULL, consumerThread, (void
*)&conArgs[i]);
        if (ret != 0)
        {
```

```
            printf("Create thread error");
            exit(1);
        }
    }

    for (i = 0; i < ProdNum; i++)
    {
        ret = pthread_create(&proid[i], NULL, productThread, (void
*)&proArgs[i]);
        if (ret != 0)
        {
            printf("Create thread error");
            exit(1);
        }
    }

    sleep((ConsNum + ProdNum) * 10);

    if (writehistorycount > readhistorycount)
        max = writehistorycount;
    else
        max = readhistorycount;

    for (i = 0; i < max; i++)
    {
        if ((i < writehistorycount) && (i < readhistorycount))
            printf("%s | %s\n", writehistory[i], readhistory[i]);
        else if (i < writehistorycount)
            printf("%s | %s\n", writehistory[i], " ");
        else
            printf("%s | %s\n", " ", readhistory[i]);
    }

    printf("*************缓冲池*************\n");
    for (i = 0; i < historycount; i++)
        printf("%s", history[i]);

    free(proid);
    free(proArgs);
    free(conid);
    free(conArgs);

    return 0;
}
```

编译链接命令：gcc pc2.c -o pc2 -lpthread

运行命令：./pc2

交互与结果：（非全部截图）

```
wuhan@wuhan-virtual-machine:~/c$ gcc pc2.c -o pc2 -lpthread
wuhan@wuhan-virtual-machine:~/c$ ./pc2
请输入生产者线程的数目 :4
请输入消费者线程的数目 :3
生产者1:缓冲区0=1 | 消费者1:缓冲区0=0

生产者2:缓冲区1=2 | 消费者3:缓冲区1=0

生产者3:缓冲区2=3 | 消费者2:缓冲区2=1

生产者4:缓冲区3=4 | 消费者3:缓冲区3=0

生产者1:缓冲区4=1 | 消费者1:缓冲区4=0

生产者2:缓冲区5=2 | 消费者2:缓冲区5=0

生产者3:缓冲区6=3 | 消费者3:缓冲区6=0

生产者4:缓冲区7=4 | 消费者1:缓冲区7=0

生产者1:缓冲区8=1 | 消费者2:缓冲区8=0

生产者2:缓冲区9=2 | 消费者1:缓冲区9=0

生产者3:缓冲区0=3 | 消费者3:缓冲区0=0

生产者4:缓冲区1=4 | 消费者2:缓冲区1=0

生产者1:缓冲区2=1 | 消费者1:缓冲区2=4

生产者2:缓冲区3=2 | 消费者3:缓冲区3=4

生产者3:缓冲区4=3 | 消费者2:缓冲区4=4

生产者4:缓冲区5=4 | 消费者3:缓冲区5=0

生产者1:缓冲区6=1 | 消费者1:缓冲区6=0

生产者2:缓冲区7=2 | 消费者2:缓冲区7=0
```

```
生产者4:缓冲区7=4 | 消费者1:缓冲区7=0

生产者1:缓冲区8=1 | 消费者3:缓冲区8=0

生产者2:缓冲区9=2 | 消费者2:缓冲区9=0

生产者3:缓冲区0=3 |
生产者4:缓冲区1=4 |
生产者1:缓冲区2=1 |
生产者2:缓冲区3=2 |
生产者3:缓冲区4=3 |
生产者4:缓冲区5=4 |
生产者1:缓冲区6=1 |
生产者2:缓冲区7=2 |
生产者3:缓冲区8=3 |
生产者4:缓冲区9=4 |
**********缓冲池**********
消费者1:缓冲区0=0
消费者3:缓冲区1=0
生产者1:缓冲区0=1
消费者2:缓冲区2=1
生产者2:缓冲区1=2
生产者3:缓冲区2=3
生产者4:缓冲区3=4
消费者3:缓冲区3=0
生产者1:缓冲区4=1
消费者1:缓冲区4=0
消费者2:缓冲区5=0
生产者2:缓冲区5=2
生产者3:缓冲区6=3
生产者4:缓冲区7=4
消费者3:缓冲区6=0
生产者1:缓冲区8=1
消费者1:缓冲区7=0
消费者2:缓冲区8=0
生产者2:缓冲区9=2
生产者3:缓冲区0=3
生产者4:缓冲区1=4
消费者1:缓冲区9=0
消费者3:缓冲区0=0
```