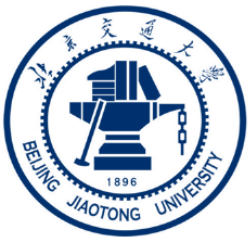

北京交通大學

BEIJING JIAOTONG UNIVERSITY

《操作系统》

典型同步问题验证性实验

实验要求

北京交通大学

1、实验目的

探索、理解并掌握操作系统同步机制的应用编程方法，针对典型的同步问题，构建基于 Windows（或 Linux）操作系统同步机制的解决方案。

2、实验内容

了解、熟悉操作系统同步机制及编程方法，针对典型同步问题的算法，譬如生产者-消费者问题、读者-写者问题、哲学家就餐问题，进行测试验证。

3、实验要求

典型同步问题包括为：从生产者-消费者问题、读者-写者问题、哲学家就餐问题等典型同步问题。

1）阅读附录中的相关代码，总结算法实现的思路。

2）测试生产者-消费者代码。验证消费者和消费者之间是否能正确的进行同步：一个生产者一个消费者、多个生产者多个消费者都能正确同步；生产者生产数据快，缓冲区满后，生产者阻塞等待；消费者消费数据快，缓冲区空后，消费者阻塞等待；缓冲区不空不满时，生产者消费者能交替运行；修改原代码，去掉同步操作命令后再编译运行，观察会出现哪些问题；修改原代码，将进入临界区的两个 wait 操作位置互换，编译后运行，观察进程死锁的现象。

3）测试哲学家进餐问题的代码。通过实验验证哲学家进餐算法会导致死锁。然后修改原代码，解决这个问题。

4）测试读者写者问题。构造五种的不同读者写者执行序列，每个序列包括 10 个读者或者写者进程，测试读者写者能否正确同步；观察读者写者的到达时间与执行时间的先后关系是否一致。

5）实现并测试写者优先的读者写者算法。（选做）

4、实验环境

开发测试环境：VS Code + Deepin

5、源码附录

producer-consumer.c 文件：

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define n 5

int in = 0, out = 0;
int buffer[n];

//empty: 资源信号量，表示缓冲池中空缓冲区的数量，其初始值为 n
//full: 资源信号量，表示缓冲池中有数据的缓冲区的数量，其初始值为 0
//mutex: 互斥信号量，实现进程对缓冲池的互斥使用
sem_t empty, full, mutex;

int id = 0;//记录产品的 id

//生产者
void *producer(void *param) {
    long ThreadId = (long)param;//线程 id
    while(1){
        sem_wait(&empty);
        sem_wait(&mutex);
        //生产产品
        buffer[in] = id;
        in = (in+1)%n;
        sleep(2);//生产产品花费时间
        printf("Thread-%ld : Producer produce product %d \n", ThreadId, id++
);
        sem_post(&mutex);
        sem_post(&full);
    }
}

//消费者
void *consumer(void *param) {
    long ThreadId = (long)param;
    while(1){
        sem_wait(&full);
        sem_wait(&mutex);
        //消费产品
```

```
40.         int item = buffer[out];
41.         out = (out+1)%n;
42.         sleep(1);//消费产品花费时间
43.         printf("Thread-%ld : Consumer consume product %d \n", ThreadId, item
    );
44.         sem_post(&mutex);
45.         sem_post(&empty);
46.     }
47. }
48.
49. int main() {
50.     //线程 id
51.     pthread_t tid[4];
52.
53.     //初始化信号量
54.     //第二个参数用于区分进程/线程间共享(0 为当前进程各线程间共享)
55.     //第三个参数为信号量初始值
56.     sem_init(&mutex, 0, 1);
57.     sem_init(&empty, 0, n);
58.     sem_init(&full, 0, 0);
59.
60.     //2 个生产者，2 个消费者
61.     pthread_create(&tid[0], NULL, consumer, (void*)0);
62.     pthread_create(&tid[1], NULL, producer, (void*)1);
63.     pthread_create(&tid[2], NULL, consumer, (void*)2);
64.     pthread_create(&tid[3], NULL, producer, (void*)3);
65.
66.     //输入 q 或 Q 结束进程
67.     while(1) {
68.         char c = getchar();
69.         if(c=='q' || c=='Q'){
70.             for(int i=0; i<4; i++){
71.                 pthread_cancel(tid[i]);
72.             }
73.             break;
74.         }
75.     }
76.
77.     //释放信号量
78.     sem_destroy(&mutex);
79.     sem_destroy(&empty);
80.     sem_destroy(&full);
81.     return 0;
82. }
```

ReaderWriter.c 文件：

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

// 互斥信号量 wmutex 用于实现对文件的互斥访问
// rmutex 用于对 readCount 变量的互斥访问
sem_t wmutex, rmutex;
//记录当前有几个读进程在访问文件
int readCount = 0;

//读者
void* Reader(void* param) {
    long threadid = (long)param;//进程 id
    while(1) {
        sem_wait(&rmutex);
        if(readCount == 0){//第一个读者到来，
            sem_wait(&wmutex);
        }
        readCount++;
        sem_post(&rmutex);
        printf("Thread-%ld: is reading...\n", threadid);
        sleep(1);
        sem_wait(&rmutex);
        readCount--;
        if(readCount == 0){//所有读者读完，释放写者
            sem_post(&wmutex);
        }
        sem_post(&rmutex);
    }
}

//写者
void* Writer(void* param) {
    long threadid = (long)param;
    while(1) {
        sem_wait(&wmutex);
        printf("Thread-%ld: is writing...\n", threadid);
        sleep(2);
        sem_post(&wmutex);
    }
```

```
}

int main() {
    //信号量初始化
    sem_init(&rmutex, 0, 1);
    sem_init(&wmutex, 0, 1);

    pthread_t tid[4];
    //两个读者、两个写者
    pthread_create(&tid[0], NULL ,Reader, (void*)0);
    pthread_create(&tid[1], NULL ,Writer, (void*)1);
    pthread_create(&tid[2], NULL ,Reader, (void*)2);
    pthread_create(&tid[3], NULL ,Writer, (void*)3);

    //输入 q 或 Q 退出
    while(1) {
        char c = getchar();
        if (c=='q' || c=='Q'){
            for (int i = 0; i < 4; ++i) {
                pthread_cancel(tid[i]);
            }
            break;
        }
    }

    //释放信号量
    sem_destroy(&rmutex);
    sem_destroy(&wmutex);
    return 0;
}
```

PhD.c 文件：

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

//mutex 互斥地取筷子
sem_t chopstick[5];

void *eat_think(void *arg)
{
    int phi = (int)arg;
    while(1) {
```

```
13.
14.         sem_wait(&chopstick[phi]);       //拿左
15.         sem_wait(&chopstick[(phi+1)%5]); //拿右
16.         printf("Philosopher %d fetches chopstick %d\n",phi,phi);
17.         printf("Philosopher %d fetches chopstick %d\n",phi,phi+1);
18.
19.         sleep(5); //吃饭
20.         printf("Philosopher %d is eating...\n",phi);
21.         sem_post(&chopstick[phi]);
22.         sem_post(&chopstick[phi+1]);
23.         printf("Philosopher %d release chopstick %d\n", phi, phi);
24.         printf("Philosopher %d release chopstick %d\n", phi, phi+1);
25.         //思考
26.         sleep(3);
27.     }
28. }
29.
30. int main() {
31.     //线程 id
32.     pthread_t tid[5];
33.
34.     for (int i = 0; i < 5; ++i) {
35.         sem_init(&chopstick[i],0,1);
36.     }
37.
38.     for (int i = 0; i < 5; ++i) {
39.         pthread_create(&tid[i] ,NULL, eat_think,(void *)i);
40.     }
41.
42.     while (1){
43.         char c = getchar();
44.         if (c=='q' || c=='Q'){
45.             for (int i = 0; i < 4; ++i) {
46.                 pthread_cancel(tid[i]);
47.             }
48.             break;
49.         }
50.     }
51.     for (int i = 0; i < 5; ++i) {
52.         sem_destroy(&chopstick[i]);
53.     }
54.
55.     return 0;
56. }
```