电子科技大学

University of Electronic Science and Technology of China

Linux操作系统编程

实验七 | 综合应用实验

主讲老师：杨珊

实验七 | 综合实验目的

目的一：了解生产者消费者问题中同步互斥的基本概念

目的二：掌握XSI信号量集机制

目的三：掌握XSI共享内存机制

目的四：掌握Linux中多进程协同工作的程序设计思想

生产者消费者问题是?

若干任务通过共享缓冲池（包含一[illegible]换数据

- ✓ “生产者”任务不断写入[illegible]而[illegible]不断读出
- ✓ 任何时[illegible]操作。

三种重要角色
1. 生产者任务
2. 消费者任务
3. 共享缓冲区

两个协同工作机制

同[illegible]产者才能写入；
有[illegible]能读出

两个协同工作机制
互斥：一个任务对缓冲池进行访问

实验七 | 生产者消费者概述

生产者：从源（文件、网络等）读取数据放入未使用的缓冲区

消费者：从已有数据的缓冲区取走数据进行处理（计算、变换等）

实验七 | 实验内容

- ✓ 综合运用所学知识在Linux系统上编写应用程序，要求至少支持2个生产者任务（进程）和2个消费者任务（进程）的同时运行
- ✓ 生产者任务和消费者任务之间通过XSI共享内存机制实现跨进程的共享缓冲池
- ✓ 生产者任务和消费者任务之间通过XSI信号量集机制实现对缓冲池的互斥访问
- ✓ 生产者与消费者之间的同步通过修改缓冲池结构中的缓冲区状态变量来实现

实验七 | 实验原理 | 缓冲区定义

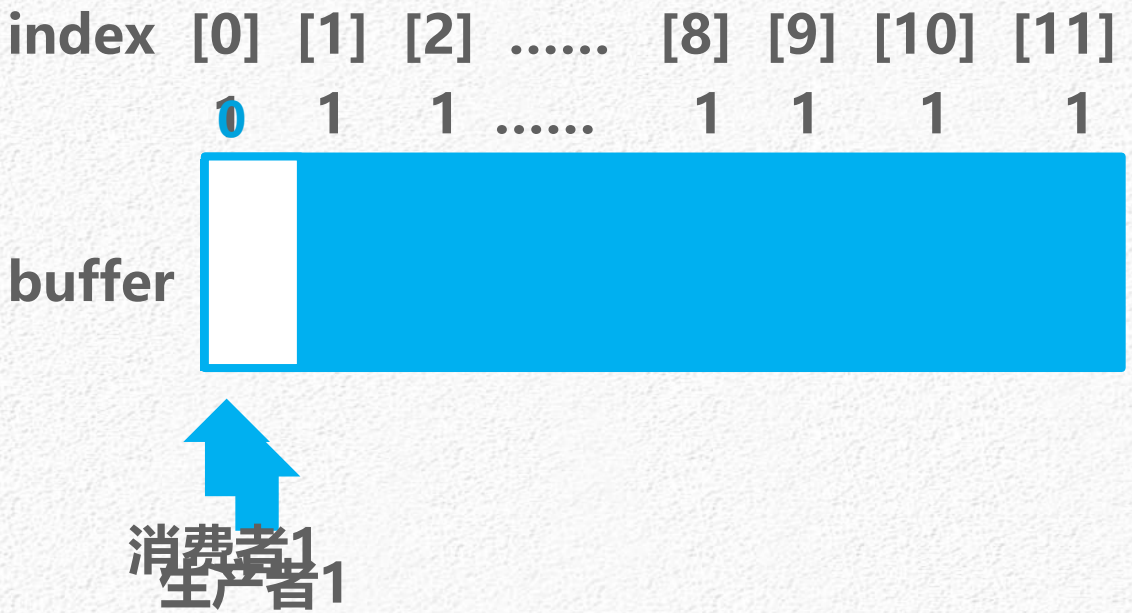
- ✓ 生产者与消费者之间的**同步**通过修改缓冲池结构中的**缓冲区状态变量**来实现

缓冲池结构（5个缓冲区）

```
struct BufferPool
{
char Buffer[5][100];   //5个缓
int Index[5];  //缓冲区状态
};
```

状态	含义
0	对应的缓冲区未被生产者使用，可分配但不可消费
1	表示对应的缓冲区以被生产者使用，不可分配但可消费

实验七 | 同步机制描述

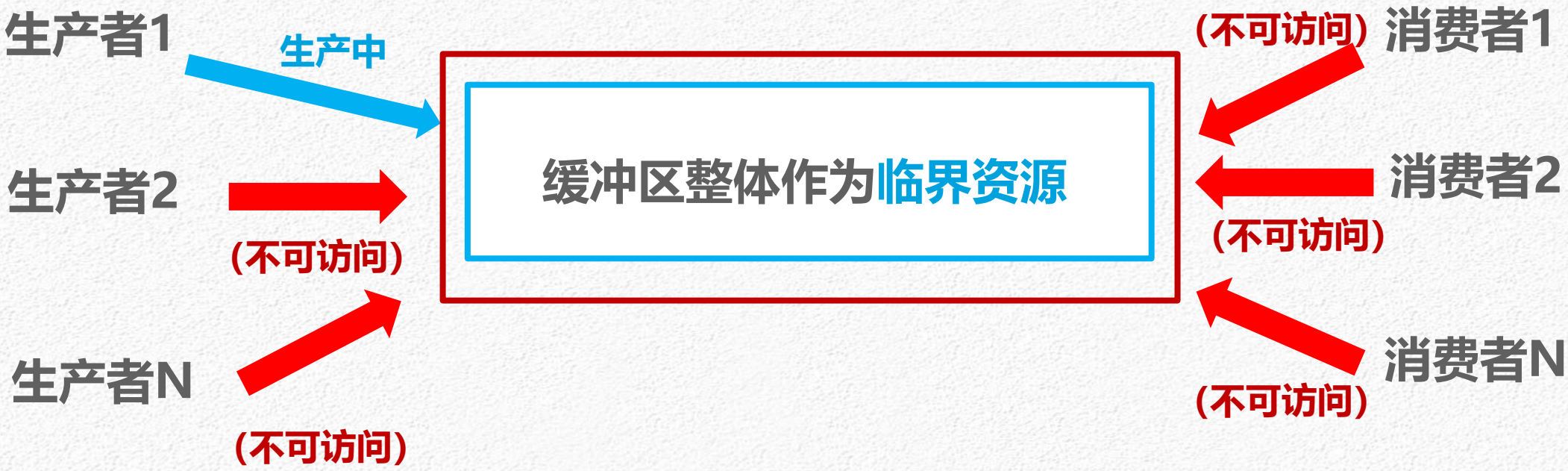


同步如何做到：

✓通过缓冲池结构中的状态变量来指示（当前实验要求）

✓通过信号量来指示未被使用缓冲区数量和已被使用缓冲区数量

实验七 | 互斥机制描述



互斥如何做到：

✓XSI信号量集（包含1个信号量，初始值为1）

✓POSIX信号量

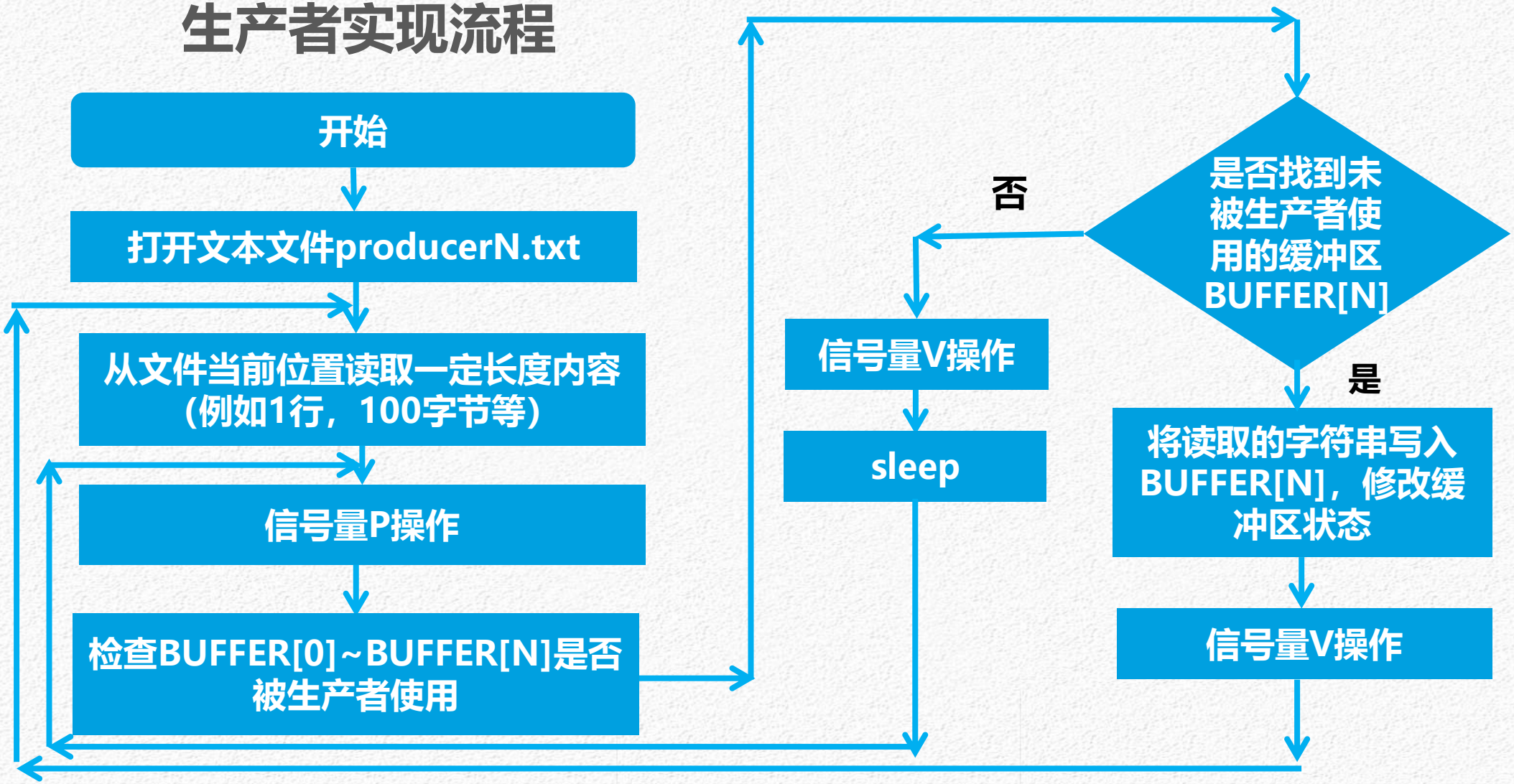
实验七 | 实验原理 | 信号量集系统调用

- ✓ 生产者任务和消费者任务之间通过XSI信号量集机制实现对缓冲池的互斥访问

功能	XSI信号量集系统调用
创建或打开信号量集对象	semget
信号量的P/V操作	semop
信号量设置初值 “删除”信号量	semctl

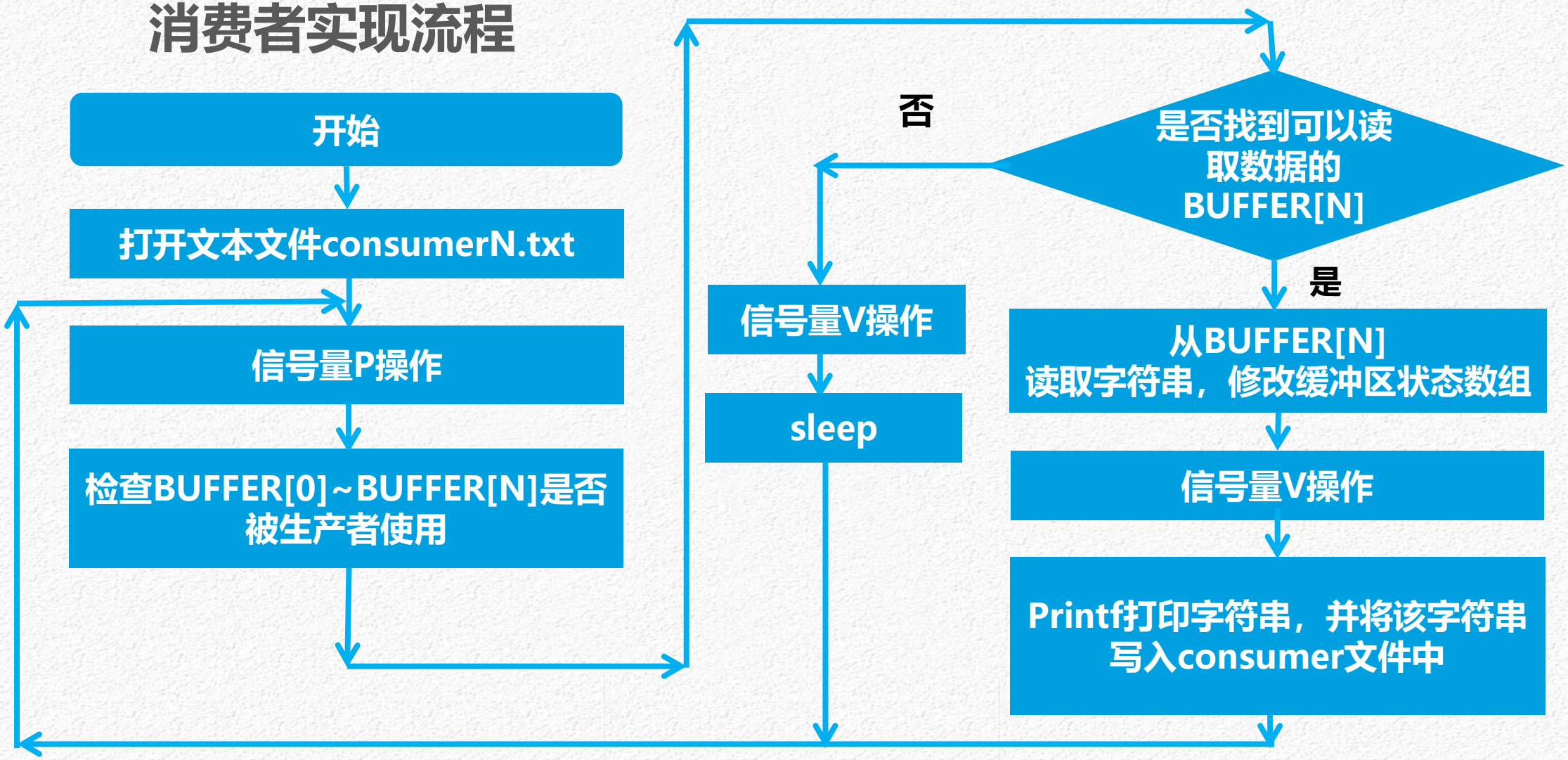
实验七｜生产者的程序实现流程

生产者实现流程



实验七｜消费者的程序实现流程

消费者实现流程



实验七｜创建或打开信号量集对象示例

```
#include<stdio.h>
#include<sys/sem.h>
#define MYKEY 0x1a0a
int main()
{
        int semid;
        semid=semget(MYKEY, 1,
                        IPC_CREAT|IPC_R | IPC_W| IPC_M);
        printf("semid=%d\n",semid);
        return 0;   }
```

实验七｜信号量设置初始值示例代码

```
int init_sem(int sem_id,init init_value)
{
    union semun sem_union;
    sem_union.val = init_value;
    if(semctl(sem_id,0,SETVAL,sem_union)==-1)
        {
            perror(“Initialize semaphore”);
            return -1;
        }
    return 1；    }
```

实验七｜创建或打开信号量集对象示例

```
int semaphore_p(int sem_id, short sem_no)
{
struct sembuf sem_b;
sem_b.sem_num = sem_no;  //信号量集中的信号量编号
sem_b.sem_op = -1;    // P操作，每次分配1个资源
sem_b.sem_flg = SEM_UNDO;
if (semop(sem_id, &sem_b, 1) == -1)
{
    perror( "semaphore_p failed");
    return 0;     }
return 1;     }
```

实验七｜创建或打开信号量集对象示例

```
int semaphore_v (int sem_id,short sem_no)
{
struct sembuf sem_b;
sem_b.sem_num = sem_no;   //信号量集中信号量编号
sem_b.sem_op = 1;                 //V操作，每次释放1个资源
sem_b.sem_flg = SEM_UNDO;
if (semop(sem_id, &sem_b, 1) == -1)
{
        perror("semaphore_v failed");
        return 0;   }
 return 1;    }
```

实验七 | 删除信号量对象示例

```
int del_sem(int sem_id)
{
    union semun sem_union;
        if(semctl(sem_id,0,IPC_RMID,sem_union)==-1)
        {
                perror(“Delete semaphore”);
                return -1;
        }
    return 1；
}
```

实验七｜实现进程间读取共享缓冲区

✓ 至少2个生产者任务（进程）和2个消费者任务（进程）的同时运行

问题：各个进程拥有自己的数据区域，如何访问struct BufferPool?

采用进程间通信的方式消息队列或者共享内存（本次实验要求）

实验七 | 实验原理 | 信号量集系统调用

- ✓ 生产者任务和消费者任务之间通过XSI共享内存机制实现跨进程共享缓冲池

功能	共享内存	消息队列
创建或打开一个IPC对象，获得对IPC的访问权	shmget	msgget
IPC操作: 发送/接收消息；连接/释放共享内存；信号量操作	shmat/ shmdt	msgsnd/ msgrcv
IPC控制：获得/修改IPC对象状态，“删除”IPC对象等	shmctl	msgctl

实验七 | 如何实现多进程同时工作

共享内存：

✓允许两个或更多进程访问同一块内存。当一个进程改变了这块内存中的内容的的时候，其他进程都会察觉到这个更改。

✓但是，系统内核**没有**对访问共享内存的进程进行同步机制

✓共享内存在各种进程间通信方式中具有最高的效率

实验七 | 共享内存相关函数

头文件：<sys/ipc.h> <sys/shm.h>

函数原型：int shmget(key_t key, size_t size, int shmflg);

作用：得到一个共享内存标识符或创建一个共享内存对象并返回共享内存标识符

- ✓ key 标识共享内存的键值： 非零键值/IPC_PRIVATE
- ✓ size 大于0的整数，新建的共享内存大小，以字节为单位，获取已存在的共享内存块标识符时，该参数为0，
- ✓ shmflg IPC_CREAT||IPC_EXCL 执行成功，保证返回一个新的共享内存标识符，附加参数指定IPC对象存储权限，如|0666

返回值：成功返回与key相关的共享内存标识符，出错返回-1，并设置error

实验七 | 共享内存相关函数

头文件：<sys/types.h> <sys/shm.h>

函数原型：int shmat(int shm_id, const void *shm_addr, int shmflg);

作用：连接共享内存标识符为shm_id的共享内存，连接成功后把共享内存区对象映射到调用进程的地址空间

- ✓ shm_id： 共享内存标识符
- ✓ shm_addr： 指定共享内存出现在进程内存地址的什么位置，通常指定为NULL，让内核自己选择一个合适的地址位置
- ✓ shmflg： SHM_RDONLY 为只读模式，其他参数为读写模式

返回值：成功返回所附加的共享内存地址，出错返回-1，并设置error

实验七 | 共享内存相关函数

头文件：<sys/types.h> <sys/shm.h>

函数原型：void *shmdt(const void* shmaddr);

作用：断开与共享内存附加点的地址，禁止本进程访问此片共享内存(不做删除操作)

- ✓ shmaddr： shmddr 连接共享内存的起始地址

返回值：成功返回0，出错返回-1，并设置error

实验七 | 共享内存相关函数

头文件：<sys/types.h> <sys/shm.h>

函数原型： int shmctl(int shmid, int cmd, struct shmid_ds* buf);

作用：控制共享内存块

- ✓ shmid：共享内存标识符
- ✓ cmd：
  - IPC_STAT：得到共享内存的状态，把共享内存的shmid_ds结构赋值到buf所指向的buf中
  - IPC_SET：改变共享内存的状态，把buf所指向的shmid_ds结构中的uid、gid、mode赋值到共享内存的shmid_ds结构内
  - IPC_RMID：删除这块共享内存
- ✓ buf：共享内存管理结构体

返回值：成功返回0，出错返回-1，并设置error

实验七｜读共享内存代码示例

```
struct sharedstruct
{
    int flag;  //非0：表示可读，0表示可写
    char content[TEXT_SZ];//记录写入和读取的字符串  };
Int main(){
    int running = 1;
    void *shm = NULL;
    struct sharedstruct *shared;
    int shmid;
    shmid = shmget((key_t)1234,
                        sizeof(struct  sharedstruct),
                        0666|IPC_CREAT);
    if(shmid == -1){
          exit(EXIT_FAILURE);}
    shm = shmat(shmid, 0, 0);
    if(shm == (void*)-1){
           exit(EXIT_FAILURE);}
    shared = (struct sharedstruct*)shm;
```

```
while(running){
     if(shared->flag != 0)
     {
          printf("You wrote: %s", shared->content);
          sleep(rand() % 3);
          shared->flag = 0;
          if(strncmp(shared->content, "end", 3) == 0)
          running= 0;
     }
     else{
           sleep(1);
     }
    if(shmdt(shm) == -1){
           exit(EXIT_FAILURE);}
    if(shmctl(shmid, IPC_RMID, 0) == -1){
           exit(EXIT_FAILURE);}
    exit(EXIT_SUCCESS);}
```

实验七｜写共享内存代码示例

```
int main()
{
    int running = 1;
    void *shm = NULL;
    struct sharedstrct*shared = NULL;
    char buffer[BUFSIZ + 1];
    int shmid;
    shmid = shmget((key_t)1234,
                   sizeof(struct sharedstruct),
                   0666|IPC_CREAT);
    if(shmid == -1){
          exit(EXIT_FAILURE);}
    shm = shmat(shmid, (void*)0, 0);
    if(shm == (void*)-1){
          exit(EXIT_FAILURE);}
    printf("Memory attached at %X\n",
              (int)shm);
    shared = (struct sharedstruct *)shm;
```

```
while(running){
    if(shared->flag == 1) {
    sleep(1);
    printf("Waiting...\n");}
    else{
    printf("Enter some text: ");
    fgets(buffer, BUFSIZ, stdin);
    strncpy(shared->content, buffer,TEXT_SZ);
    shared->flag = 1;  }
    if(strncmp(buffer, "end", 3) == 0)
          running = 0;
    }
    if(shmdt(shm) == -1) {
           exit(EXIT_FAILURE);}
    if(shmctl(shmid, IPC_RMID, 0) == -1){
           exit(EXIT_FAILURE);}
    exit(EXIT_SUCCESS);}
```

实验七 | 综合应用实验（扩展）

实验内容（扩展）

- ✓ 完善生产者和消费者程序的结束流程

例：生产者在读取完文件内容后结束自身执行，消费者在一段时间之后如果无法从缓冲池读取数据则给出提示信息，用户输入决定继续等待或退出

实验内容（选做）

- ✓ 在信号量集中支持**两个信号量**，一个信号量实现对共享缓冲区的**互斥**访问，另外一个信号量实现对缓冲区的**分配计数管理**

实验七 | 如何实现多进程同时工作

① **分别编写、编译**生产者程序和消费者程序：在shell中分别执行生产者程序或消费者程序，每执行一次产生一个生产者任务（进程）或消费者任务（进程）

② 编写**一个应用程序**，通过**fork创建子进程**，在子进程中执行生产者的代码或消费者的代码

电子科技大学

University of Electronic Science and Technology of China

Linux操作系统编程

感谢观看

主讲老师：杨珊