生产者消费者问题

1511489 李东闻

一、实验目的

实现生产者和消费者问题。

二、实验内容

模拟进程中的共享内存的实现，利用信号量同步或互斥访问共享存储区。实现生产者和消费者问题。

三、实现过程

1.了解 linux 下的信号量函数和共享内存函数

与信号量处理的函数有：semget()、semctl()、semop();分别用于创建一个新的信号量或获取一个已经存在的信号量的键值，获取控制信号量的信息，操作处理信号量。

与共享内存有关的函数有 shmget、shmat、shmdt、shmctl；分别用于得到一个共享内存标识符或创建一个共享内存对象，把共享内存区对象映射到调用进程的地址空间，断开共享内存连接，共享内存管理。

2.设计一个生产者和一个消费者共用多个缓冲区的伪代码

```
producer：
while(1)
{
P(empty); /* empty 初值为 m */
写第 in 个缓冲区; /* in 用来指示当前的第一个可写的缓冲区的下标，初值设为 0。
*/
in = (in+1)%m;
V(full);
}
consumer：
while(1)
{
```

P(full); /* full 初值为 0 */
读第 out 个缓冲区; /* out 用来指示当前的第一个可读的缓冲区的下标，初值设为 0。 */
out = (out+1)%m;
V(empty);
}

四、源代码

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>

#define PRODUCER 1 //生产者个数
#define COSTOMER 1 //消费者个数
#define WRITE_NUM 10 //写缓冲次数
#define READ_NUM 10 //读缓冲次数
#define SEM_ALL_KEY 5548
#define SEM_EMPTY 0
#define SEM_FULL 1
#define BUF_LENGTH (sizeof(struct container_buffer)) //缓冲区大小
#define BUFFER_NUM 5 //缓冲区个数
#define SHM_MODE 0600

//缓冲区结构（环）
struct container_buffer //定义共享缓冲区结构
{
    char letter[BUFFER_NUM];
    int head;
    int tail;
    int is_empty; //判断缓冲区是否为空的标志
};

//得到随机数，产生延迟时间
int random_num()
{
    int t;
    srand((unsigned)(getpid() + time(NULL)));
    t = rand() % 5;
    return t;
```

```
}

//P 操作，获得使用权
void p(int sem_id, int sem_num)
{
    struct sembuf sem_buff;
    sem_buff.sem_num = sem_num;
    sem_buff.sem_op = -1;
    sem_buff.sem_flg = 0;
    semop(sem_id, &sem_buff, 1);
}

//得到一个随机字符,模拟产品名字
char random_letter()
{
    char a;
    srand((unsigned)(getpid() + time(NULL)));
    a = (char)((char)(rand() % 15) + '!');
    return a;
}

//V 操作，释放使用权
void v(int sem_id, int sem_num)
{
    struct sembuf sem_buff;
    sem_buff.sem_num = sem_num;
    sem_buff.sem_op = 1;
    sem_buff.sem_flg = 0;
    semop(sem_id, &sem_buff, 1);
}
//主函数
int main(int argc, char * argv[])
{
    int shm_id, sem_id; //共享内存段标识变量 shm_id，信号量标识变量 sem_id
    int num_p = 0, num_c = 0, i, j; //定义生产者和消费者的个数变量，初始化为 0
    struct container_buffer * shmptr; //指向缓冲区结构的指针
    char pn; //随机字符，代表产品
    time_t now; //时间变量
    pid_t pid_p, pid_c; //进程 pid 变量
    printf("Main process starts.\n");
    sem_id = semget(SEM_ALL_KEY, 2, IPC_CREAT | 0660); //创建两个信号量, empty, full
    semctl(sem_id, SEM_EMPTY, SETVAL, BUFFER_NUM);
    //索引为 SEM_EMPTY 的信号量值为 3
    semctl(sem_id, SEM_FULL, SETVAL, 0);
```

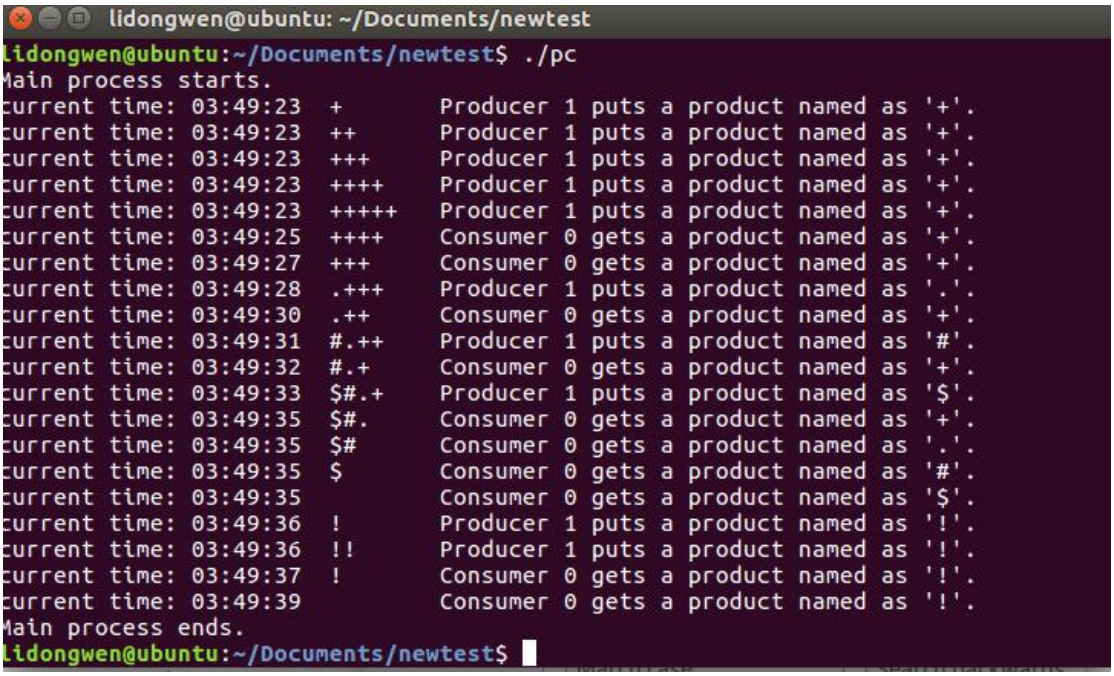
```
//索引为 SEM_FULL 的信号量值为 0
if ((shm_id = shmget(IPC_PRIVATE, BUF_LENGTH, SHM_MODE)) < 0)
   //申请一个共享主存段,大小为缓冲区大小
{
   exit(1); //失败退出
}
if ((shmptr = shmat(shm_id, 0, 0)) == (void *)-1) //将共享段与进程相连
{
   exit(1); //失败退出
}
shmptr->head = 0; //初始化缓冲区
shmptr->tail = 0;
shmptr->is_empty = 1;
while ((num_p++) < PRODUCER) //循环创建 2 个生产者
{
   if ((pid_p = fork()) < 0) //创建一个进程
   {
      exit(1); //失败退出
   }
   //如果是子进程，开始创建生产者
   if (pid_p == 0)
   {
      if ((shmptr = shmat(shm_id, 0, 0)) == (void *)-1) //将共享段与本进程相连
      {
         exit(1); //失败退出
      }
      for (i = 0; i < WRITE_NUM; i++) //循环尝试在缓冲区内放入数据为 WRITE_NUM 次
      {
         p(sem_id, SEM_EMPTY); //p 操作，申请使用权,p(empty)
         sleep(random_num()); //随机等待一段时间
         shmptr->letter[shmptr->tail] = pn = random_letter();
         //在缓冲队列里面放入一个产品
         shmptr->tail = (shmptr->tail + 1) % BUFFER_NUM;
         shmptr->is_empty = 0;
         //更新缓冲区状态为满
         now = time(NULL); //取得系统时间
         printf("current time: %02d:%02d:%02d\t",
            localtime(&now)->tm_hour, localtime(&now)->tm_min,
            localtime(&now)->tm_sec);
         for (j = (shmptr->tail - 1 >= shmptr->head) ?
            (shmptr->tail - 1) : (shmptr->tail - 1 + BUFFER_NUM); !(shmptr->is_empty)

            && j >= shmptr->head; j--)
```

```
            //输出缓冲区状态
            {
                printf("%c", shmptr->letter[j % BUFFER_NUM]);
            }
            printf("\tProducer %d puts a product named as '%c'.\n",
                num_p, pn); //输出动作序列
            fflush(stdout); //清除文件缓存区
            v(sem_id, SEM_FULL); //释放对文件的使用权，V(full)
        }
        shmdt(shmptr); //将共享段与进程之间解除链接
        exit(0); //子进程终止
    }
}
for (; num_c < COSTOMER; num_c++) //循环创建 COSTOMER 个生产者
{
    if ((pid_c = fork()) < 0) //创建一个子进程
    {
        printf("Error on fork.\n");
        exit(1); //失败退出
    }
    //如果是子进程，开始创建消费者
    if (pid_c == 0)
    {
        if ((shmptr = shmat(shm_id, 0, 0)) == (void *)-1) //将共享段与本进程相连
        {
            printf("Error on shmat.\n");
            exit(1); //失败退出
        }
        for (i = 0; i < READ_NUM; i++) //循环读 READ_NUM 次
        {
            p(sem_id, SEM_FULL); //p 操作，p(full),实现同步
            sleep(random_num()); //随机等待一段时间
            pn = shmptr->letter[shmptr->head]; //得到读取的产品标示字符
            shmptr->head = (shmptr->head + 1) % BUFFER_NUM;
            shmptr->is_empty = (shmptr->head == shmptr->tail);
            //更新缓冲区产品状态
            now = time(NULL); //得到系统时间
            printf("current time: %02d:%02d:%02d\t",
                localtime(&now)->tm_hour, localtime(&now)->tm_min,
                localtime(&now)->tm_sec);
            for (j = (shmptr->tail - 1 >= shmptr->head) ?
                (shmptr->tail - 1) : (shmptr->tail - 1 + BUFFER_NUM); !(shmptr->is_empty)
                && j >= shmptr->head; j--)
                //  输出缓冲区状态
```

```
            {
                printf("%c", shmptr->letter[j % BUFFER_NUM]);
            }
            printf("\tConsumer %d gets a product named as '%c'.\n",
                num_c, pn);
            fflush(stdout); //清除文件缓存区
            v(sem_id, SEM_EMPTY);
        }
        shmdt(shmptr); //解除共享段与本进程的连接
        exit(0);
    }
  }
  //主控程序最后退出
  while (wait(0) != -1); //等待子进程结束
  shmdt(shmptr); //结束父进程和共享段的连接
  shmctl(shm_id, IPC_RMID, 0); //删除共享内存段
  printf("Main process ends.\n");
  fflush(stdout); //清除文件缓存区
  exit(0);
}
```

五、截图



六、