On this page

实验5 同步与互斥

实验内容与任务

- 实现生产者消费者程序；生产者进程和消费者进程通过共享内存实现进程间通信交换数据。

相关知识及背景

- 生产者进程持续生产商品，在共享的仓库中存在空闲空间时，将商品存放至仓库，如此循环生产和存放商品；
- 消费者进程持续从共享的仓库中取商品消费；当仓库空时消费者需要等待生产者生产新商品并存放入仓库，才能取商品消费，如此循环持续运行；
- 当仓库满时，生产者应该等待消费者取走商品，仓库中有空闲空间才能存入新商品；
- 生产者不应该覆盖没有被使用过的商品；
- 消费都不应该重复取出已经消费过的商品；

实验过程

示例1. 生产者与消费者

1. 文件consumer.c

```
#include  <stdio.h>
#include  <stdlib.h>
#include <unistd.h>
#include  <sys/types.h>
#include  <sys/ipc.h>
#include  <sys/shm.h>

#include "utils.h"
#include <pthread.h>
#include <semaphore.h>

/* Shared Memory Global Variables */
key_t          shmKey;
int            shmId;
```

```
void          *shmPtr;
void shmInit()
{
    shmKey = ftok(".", 'x');
    shmId = shmget(shmKey, sizeof(int)*3 + sizeof(struct Product) *
BUFFER_SIZE, 0666);
    if (shmId < 0) {
        printf("*** shmget error (consumer) ***\n");
        exit(1);
    }

    printf("Consumer has received a shared memory.\n");

    shmPtr = shmat(shmId, NULL, 0);

    if ((int) shmPtr == -1) {
        printf("*** shmat error (client) ***\n");
        exit(1);
    }
}

void shmClean()
{
     shmdt((void *) shmPtr);
     printf("Consumer has detached its shared memory...\n");
     printf("Consumer exits...\n");
}

/* Shared Memory Layout */
/*
*    | counter[0] | counter[1] | flag[PRODUCER] | flag[CONSUMER] | turn |
status | Product[0] | Product[1] |......| Product[99] |
*/

void  main(void)
{
     struct Product *buffer;
     int           *counter;
     int           *status;

     int out = 0;

     struct Product next_consumed;

     shmInit();

     counter = (int*)shmPtr;
```

```
    status = counter + 2;

    buffer = (struct Product*)(status + 1);

    printf("Counter:%d\t%d\n",counter[0],counter[1]);
    printf("Consumer has attached the shared memory...\n");

    if( *status == STATUS_READY )
   *status = STATUS_RUNNING;

    while(TRUE)
    {
        while( counter[0] == 0 )
            ;
        next_consumed = buffer[ out ];
        out = (out + 1) % BUFFER_SIZE;

        counter[0]--;
        counter[1]--;

        printf("Consume the Product:\n\tId:%d\n\tName:%s\n\tValue:%d\n",
               next_consumed.Id,
               next_consumed.Name,
               next_consumed.Value);

    if(counter[0] != counter[1])
    {
            *status = STATUS_COMPLETE;
            printf("counter[0]=%d\tcounter[1]=%d\n", counter[0], counter[1]);
        break;
        }

    }

    shmClean();

    exit(0);
}
```

2. 文件producer.c

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <sys/types.h>
#include  <sys/ipc.h>
```

```
#include  <sys/shm.h>

#include <pthread.h>
#include "utils.h"
#include <semaphore.h>

/* Global Shared Memory Variables */
key_t shmKey;
int   shmId;
void *shmPtr;

void shmInit()
{
     shmKey = ftok(".", 'x');

     shmId = shmget(shmKey, sizeof(int)*3  + sizeof(struct Product) *
BUFFER_SIZE, IPC_CREAT | 0666);

     if (shmId < 0) {
          printf("*** shmget error (server) ***\n");
          exit(1);
     }

     printf("Produce create a buffer of size %d.\n", BUFFER_SIZE);

     shmPtr =  shmat(shmId, NULL, 0);
     if ((int) shmPtr == -1) {
          printf("*** shmat error (Producer) ***\n");
          exit(1);
     }
}

void shmClean()
{
     shmdt((void *) shmPtr);
     printf("Producer has detached its shared memory...\n");
     shmctl(shmId, IPC_RMID, NULL);
     printf("Producer has removed its shared memory...\n");
     printf("Producer exits...\n");
}

void  main(int  argc, char *argv[])
{

    struct Product  *buffer;
    int* counter;
    int  in = 0;
```

```
    int* status;

    struct Product product;
    int productId = 1;

    shmInit();

/* Shared Memory Layout */
/*
*    | counter[0] | counter[1] | status | Product[0] | Product[1] |......|
Product[99] |
*/
    counter = (int*)shmPtr;
    status = counter + 2;

    buffer = (struct Product*)(status + 1);

    printf("Initialize shared variables...\n");
    counter[0]=0;
    counter[1]=0;

    *status = STATUS_READY;

    while(TRUE)
    {
        if( *status == STATUS_COMPLETE )
        {
            printf("Consumer exit.\n");
            break;
        }

        /* produce a new product */
        product.Id = productId++;
        sprintf(product.Name, "N:%d",product.Id);
        product.Value = product.Id;
        printf("Produced new product %d\n", product.Id);

        while(counter[0] == BUFFER_SIZE)
            ;

        buffer[in] = product;
        in = (in + 1) % BUFFER_SIZE;

        counter[0] ++;
        counter[1] ++;
    }
```

```
    shmClean();

    exit(0);
}
```

3. utils.h文件

```
#define BUFFER_SIZE 100

#define TRUE  1
#define FALSE 0

/* Shared Memory Layout */
/*
*    | counter[0] | counter[1] | status | Product[0] | Product[1] |......|
Product[99] |
*/

/* FLAGS For communication */
#define STATUS_READY    0
#define STATUS_RUNNING  1
#define STATUS_COMPLETE 2


#define PRODUCER 0
#define CONSUMER 1

struct Product
{
    int Id;
    char Name[32];
    int Value;
};
```