计算机学院_操作系统_课程实验报告

实验题目： 实验 4 进程同步		学号：202300130183
日期：2025/4/7	班级： 23 级智能班	姓名：宋浩宇
Email：202300130183@mail.sdu.edu.cn		

实验方法介绍：

使用物理机上的 Ubuntu 24.04 来编写编译相应的代码。

实验过程描述：

首先是完成示例实验。示例实验需要先把代码全部抄写下来，但是因为作者写的代码有些问题，过不了编译，因此我们需要先修正代码里边的错误。

在 ipc.h 里有：

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#define BUFSZ 256

int get_ipc_id(char* proc_file, key_t key);
char* set_shm(key_t shm_key, int shm_num, int shm_flags);
int set_sem(key_t sem_key, int sem_val, int sem_flg);
int down(int sem_id);
int up(int sem_id);
typedef union semuns
{
  int val;
} Sem_uns;
typedef struct msgbuf
{
  long mtype;
  char mtext[1];
} Msg_buf;
extern key_t buff_key;
extern int buff_num;
extern char* buff_ptr;
extern key_t pput_key;
```

```
extern int pput_num;
extern int* pput_ptr;
extern key_t cget_key;
extern int cget_num;
extern int* cget_ptr;
extern key_t prod_key;
extern key_t pmtx_key;
extern int prod_sem;
extern int pmtx_sem;
extern key_t cons_key;
extern key_t cmtx_key;
extern int cons_sem;
extern int cmtx_sem;
extern int sem_val;
extern int sem_flg;
extern int shm_flg;
```

然后是 ipc.c 文件里的内容。

```
#include "ipc.h"
#include <sys/ipc.h>
#include <sys/sem.h>

key_t buff_key;
int buff_num;
char* buff_ptr;
key_t pput_key;
int pput_num;
int* pput_ptr;
key_t cget_key;
int cget_num;
int* cget_ptr;
key_t prod_key;
key_t pmtx_key;
int prod_sem;
int pmtx_sem;
key_t cons_key;
key_t cmtx_key;
int cons_sem;
int cmtx_sem;
int sem_val;
int sem_flg;
int shm_flg;
int get_ipc_id(char* proc_file, key_t key)
{
  FILE* pf;
```

```
  int i, j;
  char line[BUFSZ], colum[BUFSZ];
  if ((pf = fopen(proc_file, "r")) == NULL)
  {
    perror("Proc file not open");
    exit(EXIT_FAILURE);
  }
  fgets(line, BUFSZ, pf);
  while (!feof(pf))
  {
    i = j = 0;
    fgets(line, BUFSZ, pf);
    while (line[i] == ' ')
      i++;
    while (line[i] != ' ')
      colum[j++] = line[i++];
    colum[j] = '\0';
    if (atoi(colum) != key)
      continue;
    j = 0;
    while (line[i] == ' ')
      i++;
    while (line[i] != ' ')
      colum[j++] = line[i++];
    colum[j] = '\0';
    i = atoi(colum);
    fclose(pf);
    return i;
  }
  fclose(pf);
  return -1;
}
int down(int sem_id)
{
  struct sembuf buf;
  buf.sem_num = 0;
  buf.sem_op = -1;
  buf.sem_flg = SEM_UNDO;
  // buf.sem_flg = IPC_NOWAIT;
  if ((semop(sem_id, &buf, 1)) < 0)
  {
    perror("down error");
    return EXIT_FAILURE;
  }
```

```
  return EXIT_SUCCESS;
}
int up(int sem_id)
{
  struct sembuf buf;
  buf.sem_op = 1;
  buf.sem_num = 0;
  buf.sem_flg = SEM_UNDO;
  // buf.sem_flg = IPC_NOWAIT;
  if ((semop(sem_id, &buf, 1)) < 0)
  {
    // perror("up error");
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
int set_sem(key_t sem_key, int sem_val, int sem_flg)
{
  int sem_id;
  Sem_uns sem_arg;
  if ((sem_id = get_ipc_id("/proc/sysvipc/sem", sem_key)) < 0)
  {
    if ((sem_id = semget(sem_key, 1, sem_flg)) < 0)
    {
      perror("semaphore create error");
      exit(EXIT_FAILURE);
    }
    if (semctl(sem_id, 0, SETVAL, sem_arg) < 0)
    {
      perror("semaphore set error");
      exit(EXIT_FAILURE);
    }
  }
  return sem_id;
}
char* set_shm(key_t shm_key, int shm_num, int shm_flg)
{
  int i, shm_id;
  char* shm_buf;
  if ((shm_id = get_ipc_id("/proc/sysvipc/shm", shm_key)) < 0)
  {
    if ((shm_id = shmget(shm_key, shm_num, shm_flg)) < 0)
    {
      perror("shareMemory set error");
```

```
      exit(EXIT_FAILURE);
    }
    if ((shm_buf = (char*)shmat(shm_id, 0, 0)) < (char*)0)
    {
      perror("get shareMemory error");
      exit(EXIT_FAILURE);
    }
    for (i = 0; i < shm_num; i++)
      shm_buf[i] = 0;
  }
  if ((shm_buf = (char*)shmat(shm_id, 0, 0)) < (char*)0)
  {
    perror("get shareMemory error");
    exit(EXIT_FAILURE);
  }
  return shm_buf;
}
int set_msq(key_t msq_key, int msq_flg)
{
  int msq_id;
  if ((msq_id = get_ipc_id("/proc/sysvipc/msg", msq_key)) < 0)
  {
    if ((msq_id = msgget(msq_key, msq_flg)) < 0)
    {
      perror("messageQueue set error");
      exit(EXIT_FAILURE);
    }
  }
  return msq_id;
}
```

然后是 consumer.c 里的内容。

```
#include "ipc.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
  int rate;
  if (argv[1] != NULL)
    rate = atoi(argv[1]);
  else
    rate = 3;
  buff_key = 101;
  buff_num = 8;
  cget_key = 103;
```

```
  cget_num = 1;
  shm_flg = IPC_CREAT | 0644;
  buff_ptr = (char*)set_shm(buff_key, buff_num, shm_flg);
  cget_ptr = (int*)set_shm(cget_key, cget_num, shm_flg);
  prod_key = 201;
  pmtx_key = 202;
  cons_key = 301;
  cmtx_key = 302;
  sem_flg = IPC_CREAT | 0644;
  sem_val = buff_num;
  prod_sem = set_sem(prod_key, sem_val, sem_flg);
  sem_val = 0;
  cons_sem = set_sem(cons_key, sem_val, sem_flg);
  sem_val = 1;
  cmtx_sem = set_sem(cmtx_key, sem_val, sem_flg);
  // printf("set_sem success\n,prod_sem = %d,cons_sem = %d,pmtx_sem
= %d",
  // prod_sem, cons_sem, pmtx_sem);
  while (1)
  {
    fflush(stdout);
    down(cons_sem);
    down(cmtx_sem);
    sleep(rate);
    printf("%d consumerget: %c fromBuffer[%d]\n", getpid(),
buff_ptr[*cget_ptr],
           *cget_ptr);
    fflush(stdout);
    *cget_ptr = (*cget_ptr + 1) % buff_num;
    up(cmtx_sem);
    up(prod_sem);
  }
  return EXIT_SUCCESS;
}
```

然后是 producer.c 里的内容

```
#include "ipc.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
  int rate;
  if (argv[1] != NULL)
    rate = atoi(argv[1]);
  else
```

```
    rate = 3;
  buff_key = 101;
  buff_num = 8;
  pput_key = 102;
  pput_num = 1;
  shm_flg = IPC_CREAT | 0644;
  buff_ptr = (char*)set_shm(buff_key, buff_num, shm_flg);
  // printf("buff_ptr set success\n");
  pput_ptr = (int*)set_shm(pput_key, pput_num, shm_flg);
  // printf("pput_ptr set success\n");
  prod_key = 201;
  pmtx_key = 202;
  cons_key = 301;
  cmtx_key = 302;
  sem_flg = IPC_CREAT | 0644;
  prod_sem = set_sem(prod_key, sem_val, sem_flg);
  sem_val = 0;
  cons_sem = set_sem(cons_key, sem_val, sem_flg);
  sem_val = 1;
  pmtx_sem = set_sem(pmtx_key, sem_val, sem_flg);
  // printf("set_sem success,prod_sem = %d,cons_sem = %d,pmtx_sem
= %d\n",
  // prod_sem, cons_sem, pmtx_sem);
  // fflush(stdout);
  while (1)
  {
    down(prod_sem);
    down(pmtx_sem);
    buff_ptr[*pput_ptr] = 'A' + *pput_ptr;
    sleep(rate);
    printf("%d producer put: %c to Buffer[%d]\n", getpid(),
buff_ptr[*pput_ptr],
           *pput_ptr);
    fflush(stdout);
    *pput_ptr = (*pput_ptr + 1) % buff_num;
    up(pmtx_sem);
    up(cons_sem);
  }
  return EXIT_SUCCESS;
}
```

然后是 Makefile 里的内容

```
hdrs = ipc.h
opts = -g -c
c_src = consumer.c ipc.c
```

```
c_obj = consumer.o ipc.o
p_src = producer.c ipc.c
p_obj = producer.o ipc.o
all: consumer producer
consumer:$(c_obj)
    gcc $(c_obj) -o consumer
consumer.o:$(c_src) $(hdrs)
    gcc $(opts) $(c_src)
producer: $(p_obj)
    gcc $(p_obj) -o producer
producer.o:$(p_src) $(hdrs)
    gcc $(opts) $(p_src)
clean:
    rm consumer producer *.o
```

最后是实验结果截图：
只运行一个生产者：

```
25520 producer put: A to Buffer[0]
25520 producer put: B to Buffer[1]
25520 producer put: C to Buffer[2]
25520 producer put: D to Buffer[3]
25520 producer put: E to Buffer[4]
25520 producer put: F to Buffer[5]
25520 producer put: G to Buffer[6]
25520 producer put: H to Buffer[7]
```

再运行一个生产者
程序会阻塞没有输出
然后是带着消费者的

```
25520 producer put: A to Buffer[0]      27063,consumerget:AfromBuffer[0]      27345,consumerget:EfromBuffer[4]
25520 producer put: B to Buffer[1]      27063,consumerget:BfromBuffer[1]      27345,consumerget:GfromBuffer[6]
25520 producer put: C to Buffer[2]      27063,consumerget:CfromBuffer[2]      27345,consumerget:AfromBuffer[0]
25520 producer put: D to Buffer[3]      27063,consumerget:DfromBuffer[3]      27345,consumerget:CfromBuffer[2]
25520 producer put: E to Buffer[4]      27063,consumerget:EfromBuffer[4]
25520 producer put: F to Buffer[5]      27063,consumerget:FfromBuffer[5]
25520 producer put: G to Buffer[6]      27063,consumerget:GfromBuffer[6]
25520 producer put: H to Buffer[7]      27063,consumerget:HfromBuffer[7]
25520 producer put: A to Buffer[0]      27063,consumerget:AfromBuffer[0]
25520 producer put: C to Buffer[2]      27063,consumerget:BfromBuffer[1]
25520 producer put: E to Buffer[4]      27063,consumerget:CfromBuffer[2]
25520 producer put: G to Buffer[6]      27063,consumerget:DfromBuffer[3]
25520 producer put: A to Buffer[0]      27063,consumerget:FfromBuffer[5]
25520 producer put: C to Buffer[2]      27063,consumerget:HfromBuffer[7]
25520 producer put: E to Buffer[4]      27063,consumerget:BfromBuffer[1]
25520 producer put: G to Buffer[6]      27063,consumerget:DfromBuffer[3]
25520 producer put: A to Buffer[0]
25520 producer put: C to Buffer[2]
```

然后我们需要完成独立实验，我们会在独立实验里修正一些示例实验里的缺陷。
先是消费者生产者公用的 common. h

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

```c
#include <string.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
#define SHM_SIZE 1024
#define SHM_MODE 0666 | IPC_CREAT
#define SEM_MODE 0666 | IPC_CREAT
#define SEM_VALUE 1
#define DIVIDER 20
#define PATH "common.h"
#define SLEEP_TIME 1 // 1s
key_t key_for_paper;
key_t key_for_water;
key_t key_for_tobacco;
key_t key_for_producer[3];
key_t key_for_consumer[3];
key_t key_for_paper_cnt;
key_t key_for_water_cnt;
key_t key_for_tobacco_cnt;
int shmids[3] = {0, 0, 0};
void* shm_ptrs[3];
int semids_for_producer[3]; // 表明是否有生产者正在生产
int semids_for_consumer[3]; // 表明是否有消费者正在消费
int semid_for_paper;        // 表明待生产的纸张数量
int semid_for_water;        // 表明待生产的水的数量
int semid_for_tobacco;      // 表明待生产的烟草的数量
int semids_for_paper_cnt;   // 表明纸张的数量
int semids_for_water_cnt;   // 表明水的数量
int semids_for_tobacco_cnt; // 表明烟草的数量
size_t paper_cnt;           // 表明生产总量
size_t water_cnt;           // 表明生产总量
size_t tobacco_cnt;         // 表明生产总量
static inline int createSharedMemory(key_t key, int size)
{
  int shm_id = shmget(key, size, SHM_MODE);
  if (shm_id == -1)
  {
    perror("shmget");
    return -1;
  }
  return shm_id;
```

```
}
static inline int getSharedMemory(key_t key, int size)
{
  int shm_id = shmget(key, size, SHM_MODE);
  if (shm_id == -1)
  {
    perror("shmget");
    return -1;
  }
  return shm_id;
}

static inline int iniSharedMemoryID()
{
  shmids[0] = createSharedMemory(key_for_paper, SHM_SIZE);
  shmids[1] = createSharedMemory(key_for_water, SHM_SIZE);
  shmids[2] = createSharedMemory(key_for_tobacco, SHM_SIZE);
  if (shmids[0] == -1 || shmids[1] == -1 || shmids[2] == -1)
  {
    perror("shmget");
    return -1;
  }
  return 0;
}
static inline int getKeys()
{
  key_for_paper = ftok(PATH, 'p');
  key_for_water = ftok(PATH, 'w');
  key_for_tobacco = ftok(PATH, 't');
  if (key_for_paper == -1 || key_for_water == -1 || key_for_tobacco ==
-1)
  {
    perror("ftok");
    return -1;
  }
  return 0;
}
static inline int matSharedMemory()
{
  shm_ptrs[0] = shmat(shmids[0], NULL, 0);
  shm_ptrs[1] = shmat(shmids[1], NULL, 0);
  shm_ptrs[2] = shmat(shmids[2], NULL, 0);
  if (shm_ptrs[0] == (void*)-1 || shm_ptrs[1] == (void*)-1 ||
      shm_ptrs[2] == (void*)-1)
```

```
  {
    perror("shmat");
    return -1;
  }
  return 0;
}
static inline int iniSharedMemory()
{
  if (shmids[0] == 0 || shmids[1] == 0 || shmids[2] == 0)
  {
    if (getKeys() == -1)
    {
      perror("ftok");
      return -1;
    }
  }
  else
  {
    printf("Shared memory already exists.\n");
    return -1;
  }
  if (iniSharedMemoryID() == -1)
  {
    perror("shmget");
    return -1;
  }
  if (matSharedMemory() == -1)
  {
    perror("shmat");
    return -1;
  }
  return 0;
}
static inline int writeSharedMemory(void* shm_ptr, const void* data,
                                    size_t size)
{
  if (size > SHM_SIZE)
  {
    return -1;
  }
  if (shm_ptr == NULL || shm_ptr == (void*)-1 || data == NULL)
  {
    return -1;
  }
```

```
  memcpy(shm_ptr, data, size);
  return 0;
}
static inline int writeForPaper(const void* data, size_t size)
{
  return writeSharedMemory(shm_ptrs[0], data, size);
}
static inline int writeForWater(const void* data, size_t size)
{
  return writeSharedMemory(shm_ptrs[1], data, size);
}
static inline int writeForTobacco(const void* data, size_t size)
{
  return writeSharedMemory(shm_ptrs[2], data, size);
}
static inline int readSharedMemory(void* shm_ptr, void* data, size_t
size)
{
  if (size > SHM_SIZE)
  {
    return -1;
  }
  if (shm_ptr == NULL || shm_ptr == (void*)-1 || data == NULL)
  {
    return -1;
  }
  memcpy(data, shm_ptr, size);
  return 0;
}
static inline int readForPaper(void* data, size_t size)
{
  return readSharedMemory(shm_ptrs[0], data, size);
}
static inline int readForWater(void* data, size_t size)
{
  return readSharedMemory(shm_ptrs[1], data, size);
}
static inline int readForTobacco(void* data, size_t size)
{
  return readSharedMemory(shm_ptrs[2], data, size);
}
static inline int freeSharedMemory(int shm_id)
{
  if (shmctl(shm_id, IPC_STAT, 0) != 1)
```

```
  {
    return 0;
  }
  if (shmctl(shm_id, IPC_RMID, 0) == -1)
  {
    perror("shmctl");
    return -1;
  }
  return 0;
}
static inline int dematSharedMemory()
{
  if (shmdt(shm_ptrs[0]) == -1 || shmdt(shm_ptrs[1]) == -1 ||
      shmdt(shm_ptrs[2]) == -1)
  {
    perror("shmdt");
    return -1;
  }
  return 0;
}
static inline int freeAllSharedMemory()
{
  if (dematSharedMemory() == -1)
  {
    perror("shmdt");
    return -1;
  }
  if (freeSharedMemory(shmids[0]) == -1 || freeSharedMemory(shmids[1])
== -1 ||
      freeSharedMemory(shmids[2]) == -1)
  {
    perror("shmctl");
    return -1;
  }
  return 0;
}
static inline int getSemaphoreKey()
{
  key_for_producer[0] = ftok(PATH, 'P');
  key_for_producer[1] = ftok(PATH, 'W');
  key_for_producer[2] = ftok(PATH, 'T');
  key_for_consumer[0] = ftok(PATH, 'p' + 'P');
  key_for_consumer[1] = ftok(PATH, 'w' + 'W');
  key_for_consumer[2] = ftok(PATH, 't' + 'T');
```

```
  key_for_paper_cnt = ftok(PATH, 'c' + 'P');
  key_for_water_cnt = ftok(PATH, 'c' + 'W');
  key_for_tobacco_cnt = ftok(PATH, 'c' + 'T');
  if (key_for_producer[0] == -1 || key_for_producer[1] == -1 ||
      key_for_producer[2] == -1)
  {
    perror("ftok");
    return -1;
  }
  return 0;
}
static inline int getSemaphore(key_t key)
{
  int sem_id = semget(key, SEM_VALUE, SEM_MODE);
  if (sem_id == -1)
  {
    perror("semget");
    return -1;
  }
  return sem_id;
}
static inline int down(int sem_id)
{
  struct sembuf sem[1];
  sem[0].sem_num = 0;
  sem[0].sem_op = -1;
  sem[0].sem_flg = SEM_UNDO;
  if (semop(sem_id, sem, 1) == -1)
  {
    perror("semop");
    return -1;
  }
  return 0;
}
static inline int up(int sem_id)
{
  struct sembuf sem[1];
  sem[0].sem_num = 0;
  sem[0].sem_op = 1;
  sem[0].sem_flg = SEM_UNDO;
  if (semop(sem_id, sem, 1) == -1)
  {
    perror("semop");
    return -1;
```

```
  }
  return 0;
}
static inline int iniSemaphore()
{
  if (semids_for_producer[0] == 0 || semids_for_producer[1] == 0 ||
      semids_for_producer[2] == 0 || semids_for_consumer[0] == 0 ||
      semids_for_consumer[1] == 0 || semids_for_consumer[2] == 0 ||
      semids_for_paper_cnt == 0 || semids_for_water_cnt == 0 ||
      semids_for_tobacco_cnt == 0 || semid_for_paper == 0 ||
      semid_for_water == 0 || semid_for_tobacco == 0)
  {
    if (getKeys() == -1)
    {
      perror("ftok");
      return -1;
    }
  }
  else
  {
    printf("Semaphore already exists.\n");
    return -1;
  }
  if (getSemaphoreKey() == -1)
  {
    perror("ftok");
    return -1;
  }
  semids_for_producer[0] = getSemaphore(key_for_producer[0]);
  semids_for_producer[1] = getSemaphore(key_for_producer[1]);
  semids_for_producer[2] = getSemaphore(key_for_producer[2]);
  semid_for_paper = getSemaphore(key_for_paper);
  semid_for_water = getSemaphore(key_for_water);
  semid_for_tobacco = getSemaphore(key_for_tobacco);
  semids_for_consumer[0] = getSemaphore(key_for_consumer[0]);
  semids_for_consumer[1] = getSemaphore(key_for_consumer[1]);
  semids_for_consumer[2] = getSemaphore(key_for_consumer[2]);
  semids_for_paper_cnt = getSemaphore(key_for_paper_cnt);
  semids_for_water_cnt = getSemaphore(key_for_water_cnt);
  semids_for_tobacco_cnt = getSemaphore(key_for_tobacco_cnt);
  if (semids_for_producer[0] == -1 || semids_for_producer[1] == -1 ||
      semids_for_producer[2] == -1)
  {
    perror("semget");
```

```
    return -1;
  }
  if (semid_for_paper == -1 || semid_for_water == -1 || semid_for_tobacco
== -1)
  {
    perror("semget");
    return -1;
  }
  if (semids_for_consumer[0] == -1 || semids_for_consumer[1] == -1 ||
      semids_for_consumer[2] == -1)
  {
    perror("semget");
    return -1;
  }
  if (semids_for_paper_cnt == -1 || semids_for_water_cnt == -1 ||
      semids_for_tobacco_cnt == -1)
  {
    perror("semget");
    return -1;
  }
  // 初始化生产者
  up(semids_for_producer[0]);
  up(semids_for_producer[1]);
  up(semids_for_producer[2]);
  // 初始化消费者
  up(semids_for_consumer[0]);
  up(semids_for_consumer[1]);
  up(semids_for_consumer[2]);
  return 0;
}
static inline int freeSemaphore(int sem_id)
{
  if (semctl(sem_id, 0, IPC_STAT, 0) != 1)
  {
    return 0;
  }
  if (semctl(sem_id, 0, IPC_RMID, 0) == -1)
  {
    perror("semctl");
    return -1;
  }
  return 0;
}
static inline int freeAllSemaphore()
```

```
{
  if (freeSemaphore(semids_for_producer[0]) == -1 ||
      freeSemaphore(semids_for_producer[1]) == -1 ||
      freeSemaphore(semids_for_producer[2]) == -1)
  {
    perror("semctl");
    return -1;
  }
  if (freeSemaphore(semid_for_paper) == -1 ||
      freeSemaphore(semid_for_water) == -1 ||
      freeSemaphore(semid_for_tobacco) == -1)
  {
    perror("semctl");
    return -1;
  }
  if (freeSemaphore(semids_for_consumer[0]) == -1 ||
      freeSemaphore(semids_for_consumer[1]) == -1 ||
      freeSemaphore(semids_for_consumer[2]) == -1)
  {
    perror("semctl");
    return -1;
  }
  if (freeSemaphore(semids_for_paper_cnt) == -1 ||
      freeSemaphore(semids_for_water_cnt) == -1 ||
      freeSemaphore(semids_for_tobacco_cnt) == -1)
  {
    perror("semctl");
    return -1;
  }
  return 0;
}
static inline int produce(int type, void* shm_ptr, void* data, int size)
{
  // 如果有生产者正在生产，则等待
  if (down(semids_for_producer[type]) == -1)
  {
    perror("semop");
    return -1;
  }
  // 如果没有生产任务，则等待
  if (type == 0)
  {
    down(semid_for_paper);
  }
```

```
  else if (type == 1)
  {
    down(semid_for_water);
  }
  else if (type == 2)
  {
    down(semid_for_tobacco);
  }
  if (writeSharedMemory(shm_ptr, data, size) == -1)
  {
    perror("writeSharedMemory");
    return -1;
  }
  // 生产完成，通知其他生产者和消费者
  if (up(semids_for_producer[type]) == -1)
  {
    perror("semop");
    return -1;
  }
  return 0;
}
static inline int producePaper()
{
  if (produce(0, shm_ptrs[0], (void*)&paper_cnt, sizeof(size_t)) == -1)
  {
    perror("producePaper");
    return -1;
  }
  paper_cnt++;
  // 生产完成，通知消费者
  up(semids_for_paper_cnt);
  return 0;
}
static inline int produceWater()
{
  if (produce(1, shm_ptrs[1], (void*)&water_cnt, sizeof(size_t)) == -1)
  {
    perror("produceWater");
    return -1;
  }
  water_cnt++;
  // 生产完成，通知消费者
  up(semids_for_water_cnt);
  return 0;
```

```
}
static inline int produceTobacco()
{
  if (produce(2, shm_ptrs[2], (void*)&tobacco_cnt, sizeof(size_t)) ==
-1)
  {
    perror("produceTobacco");
    return -1;
  }
  tobacco_cnt++;
  // 生产完成，通知消费者
  up(semids_for_tobacco_cnt);
  return 0;
}
static inline int consumePaper(void* data_buf, size_t size)
{
  // 如果有消费者正在消费，则等待
  down(semids_for_consumer[0]);
  // 呼起生产者
  up(semid_for_paper);
  // 等待生产者完成生产
  down(semids_for_paper_cnt);
  if (readSharedMemory(shm_ptrs[0], data_buf, size) == -1)
  {
    perror("readSharedMemory");
    return -1;
  }
  // 唤起其他消费者
  up(semids_for_consumer[0]);
  return 0;
}
static inline int consumeWater(void* data_buf, size_t size)
{
  // 如果有消费者正在消费，则等待
  down(semids_for_consumer[1]);
  // 呼起生产者
  up(semid_for_water);
  // 等待生产者完成生产
  down(semids_for_water_cnt);
  if (readSharedMemory(shm_ptrs[1], data_buf, size) == -1)
  {
    perror("readSharedMemory");
    return -1;
  }
```

```
  // 唤起其他消费者
  up(semids_for_consumer[1]);
  return 0;
}
static inline int consumeTobacco(void* data_buf, size_t size)
{
  // 如果有消费者正在消费，则等待
  down(semids_for_consumer[2]);
  // 呼起生产者
  up(semid_for_tobacco);
  // 等待生产者完成生产
  down(semids_for_tobacco_cnt);
  if (readSharedMemory(shm_ptrs[2], data_buf, size) == -1)
  {
    perror("readSharedMemory");
    return -1;
  }
  // 唤起其他消费者
  up(semids_for_consumer[2]);
  return 0;
}
static inline int produceLoop(int type)
{
  time_t current_time;
  time(&current_time);
  struct tm* time_info = localtime(&current_time);
  char time_str[20];
  strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
  if (type == 0)
  {
    producePaper();
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    printf("[ %s ] Producer Pid : %d, produce paper. paper_cnt : %d\n",
           time_str, getpid(), (int)paper_cnt);
    produceWater();
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    printf("[ %s ] Producer Pid : %d, produce water. water_cnt : %d\n",
```

```
          time_str, getpid(), (int)water_cnt);
  }
  if (type == 1)
  {
    produceWater();
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    printf("[ %s ] Producer Pid : %d, produce water. water_cnt : %d\n",
          time_str, getpid(), (int)water_cnt);
    produceTobacco();
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    printf("[ %s ] Producer Pid : %d, produce tobacco.
tobacco_cnt : %d\n",
          time_str, getpid(), (int)tobacco_cnt);
  }
  if (type == 2)
  {
    produceTobacco();
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    printf("[ %s ] Producer Pid : %d, produce tobacco.
tobacco_cnt : %d\n",
          time_str, getpid(), (int)tobacco_cnt);
    producePaper();
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    printf("[ %s ] Producer Pid : %d, produce paper. paper_cnt : %d\n",
          time_str, getpid(), (int)paper_cnt);
  }
  fflush(stdout);
  return 0;
}
static inline int producerRun()
{
  while (1)
```

```
  {
    produceLoop(0);
    produceLoop(1);
    produceLoop(2);
    sleep(SLEEP_TIME);
  }
  return 0;
}
static inline int consumeLoop(int type, void* data_buf, size_t size)
{
  time_t current_time;
  time(&current_time);
  struct tm* time_info = localtime(&current_time);
  char time_str[20];
  strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
  if (type == 0)
  {
    consumePaper(data_buf, size);
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    printf("[ %s ] Consumer Pid : %d, consume paper. got data : %d\n",
time_str,
           getpid(), *(int*)data_buf);
    consumeWater(data_buf, size);
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    printf("[ %s ] Consumer Pid : %d, consume water. got data : %d\n",
time_str,
           getpid(), *(int*)data_buf);
  }
  else if (type == 1)
  {
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    consumeWater(data_buf, size);
    printf("[ %s ] Consumer Pid : %d, consume water. got data : %d\n",
time_str,
```

```
            getpid(), *(int*)data_buf);
    consumeTobacco(data_buf, size);
    time(&current_time);
    time_info = localtime(&current_time);
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S",
time_info);
    printf("[ %s ] Consumer Pid : %d, consume tobacco. got data : %d\n",
           time_str, getpid(), *(int*)data_buf);
  }
  else if (type == 2)
  {
    consumeTobacco(data_buf, size);
    printf("[ %s ] Consumer Pid : %d, consume tobacco. got data : %d\n",
           time_str, getpid(), *(int*)data_buf);
    consumePaper(data_buf, size);
    printf("[ %s ] Consumer Pid : %d, consume paper. got data : %d\n",
time_str,
           getpid(), *(int*)data_buf);
  }
  fflush(stdout);
  return 0;
}
static inline int consumerRun(int type)
{
  int data_buf[1];
  while (1)
  {
    consumeLoop(type, data_buf, sizeof(data_buf));
    sleep(SLEEP_TIME);
  }
  return 0;
}
```

然后是 consumer.c

```
#include "common.h"

int run(int type)
{
  // if (shm_ptrs[0] == 0 || shm_ptrs[1] == 0 || shm_ptrs[2] == 0)
  // {
  //   printf("共享内存未初始化！");
  //   return -1;
  // }
  iniSharedMemory();
  iniSemaphore();
```

```
  // char buf[1];
  // readForPaper(buf, sizeof(buf));
  // printf("从共享内存中接收到了字符: %c\n", buf[0]);
  consumerRun(type);
  return 0;
}
int main(int argc, char* argv[])
{
  int type = 1;
  for (int i = 1; i < argc; i++)
  {
    if (strcmp(argv[i], "-t") == 0)
    {
      type = atoi(argv[i + 1]);
    }
    // if (strcmp(argv[i], "-i") == 0)
    // {
    //
    // }
  }
  run(type);
  return 0;
}
```

还有 producer.c

```
#include "common.h"

int run()
{
  if (iniSharedMemory() == -1)
  {
    perror("初始化共享内存失败");
    return -1;
  }
  if (iniSemaphore() == -1)
  {
    perror("初始化信号量失败");
    freeAllSharedMemory();
    return -1;
  }
  producerRun();
  freeAllSharedMemory();
  freeAllSemaphore();
  return 0;
}
```

```
int main(int argc, char const* argv[])
{
  int timer = 1;
  for (int i = 1; i < argc; i++)
  {
    if (strcmp(argv[i], "-t") == 0)
    {
      timer = atoi(argv[i + 1]);
    }
    // if (strcmp(argv[i], "-i") == 0)
    // {
    //
    // }
  }
  run();
  return 0;
}
```

然后是构建使用的 Makefile 文件：

```
hdrs = common.h
opts = -g -c
p_src = producer.c
p_obj = producer.o
c_src = consumer.c
c_obj = consumer.o
all: producer consumer
producer: $(p_obj)
    gcc $(p_obj) -o producer
producer.o:$(p_src) $(hdrs)
    gcc $(opts) $(p_src)
consumer.o:$(c_src) $(hdrs)
    gcc $(opts) $(c_src)
consumer: $(c_obj)
    gcc $(c_obj) -o consumer
clean:
    rm producer consumer *.o
```

然后为了让测试的结果更直观完整一些，并且消除手动调用这几个线程的时间误差问题，我们编写了 test.sh 文件来启动测试，并将程序输出重定向到 log.txt 文件中记录程序运行的日志。test.sh 文件中的内容为：

```
producer_process_name="producer"
if [ -f log.txt ] ; then
  echo "log.txt already exists. Do you want to overwrite it?(y/n)"
    read answer
    if [ $"answer" = "y" ] ; then
      rm log.txt
```

```
    fi
fi
touch log.txt
# if ! pgrep "$producer_process_name" >> ./log.txt ; then
#   ./producer 1 > /dev/null &
# fi
./producer >> ./log.txt &
./producer >> ./log.txt &
./consumer -t 0 >> ./log.txt &
./consumer -t 1 >> ./log.txt &
./consumer -t 2 >> ./log.txt &
echo "All processes started successfully\n"
ps | grep producer
ps | grep consumer
```

然后也是为了我们停止这个实验能够更方便快捷消除手动的时间误差，我们还有一个停止测试的脚本 endtest.sh 文件：

```
if ! ps | grep -q producer || ! ps | grep -q consumer; then
  echo "No producer or consumer process found."
  exit 1
fi
echo "these following processes will be killed:\n"

ps | grep producer
ps | grep consumer
pkill -f "producer"
pkill -f "consumer"
echo "Done."
```

这两个.sh 文件使用的是大部分 shell 兼容的语法，我实际使用的是 zsh，但经过测试 bash 也是可以使用的。

最后是结果，我们让程序运行了一段时间（约 10s）：

```
[ 2025-04-13 21:49:35 ] Producer Pid : 90133, produce paper. paper_cnt : 1
[ 2025-04-13 21:49:35 ] Producer Pid : 90133, produce water. water_cnt : 1
[ 2025-04-13 21:49:35 ] Consumer Pid : 90134, consume paper. got data : 0
[ 2025-04-13 21:49:35 ] Consumer Pid : 90134, consume water. got data : 1
[ 2025-04-13 21:49:35 ] Producer Pid : 90133, produce water. water_cnt : 2
[ 2025-04-13 21:49:35 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
1
[ 2025-04-13 21:49:35 ] Consumer Pid : 90135, consume water. got data : 0
[ 2025-04-13 21:49:35 ] Consumer Pid : 90135, consume tobacco. got data : 1
[ 2025-04-13 21:49:35 ] Consumer Pid : 90136, consume tobacco. got data : 0
[ 2025-04-13 21:49:35 ] Consumer Pid : 90136, consume paper. got data : 0
[ 2025-04-13 21:49:35 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
2
```

```
[ 2025-04-13 21:49:36 ] Producer Pid : 90133, produce paper. paper_cnt : 2
[ 2025-04-13 21:49:35 ] Producer Pid : 90132, produce paper. paper_cnt : 1
[ 2025-04-13 21:49:36 ] Producer Pid : 90132, produce water. water_cnt : 1
[ 2025-04-13 21:49:36 ] Producer Pid : 90132, produce water. water_cnt : 2
[ 2025-04-13 21:49:36 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
1
[ 2025-04-13 21:49:36 ] Consumer Pid : 90134, consume paper. got data : 1
[ 2025-04-13 21:49:36 ] Consumer Pid : 90134, consume water. got data : 1
[ 2025-04-13 21:49:36 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
2
[ 2025-04-13 21:49:36 ] Producer Pid : 90132, produce paper. paper_cnt : 2
[ 2025-04-13 21:49:36 ] Consumer Pid : 90135, consume water. got data : 1
[ 2025-04-13 21:49:36 ] Consumer Pid : 90135, consume tobacco. got data : 1
[ 2025-04-13 21:49:36 ] Consumer Pid : 90136, consume tobacco. got data : 0
[ 2025-04-13 21:49:36 ] Consumer Pid : 90136, consume paper. got data : 1
[ 2025-04-13 21:49:37 ] Consumer Pid : 90134, consume paper. got data : 2
[ 2025-04-13 21:49:37 ] Consumer Pid : 90134, consume water. got data : 2
[ 2025-04-13 21:49:37 ] Producer Pid : 90133, produce paper. paper_cnt : 3
[ 2025-04-13 21:49:37 ] Producer Pid : 90133, produce water. water_cnt : 3
[ 2025-04-13 21:49:37 ] Producer Pid : 90133, produce water. water_cnt : 4
[ 2025-04-13 21:49:37 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
3
[ 2025-04-13 21:49:37 ] Consumer Pid : 90135, consume water. got data : 3
[ 2025-04-13 21:49:37 ] Consumer Pid : 90135, consume tobacco. got data : 3
[ 2025-04-13 21:49:37 ] Consumer Pid : 90136, consume tobacco. got data : 2
[ 2025-04-13 21:49:37 ] Consumer Pid : 90136, consume paper. got data : 2
[ 2025-04-13 21:49:37 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
4
[ 2025-04-13 21:49:38 ] Producer Pid : 90133, produce paper. paper_cnt : 4
[ 2025-04-13 21:49:37 ] Producer Pid : 90132, produce paper. paper_cnt : 3
[ 2025-04-13 21:49:38 ] Producer Pid : 90132, produce water. water_cnt : 3
[ 2025-04-13 21:49:38 ] Producer Pid : 90132, produce water. water_cnt : 4
[ 2025-04-13 21:49:38 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
3
[ 2025-04-13 21:49:38 ] Consumer Pid : 90134, consume paper. got data : 3
[ 2025-04-13 21:49:38 ] Consumer Pid : 90134, consume water. got data : 2
[ 2025-04-13 21:49:38 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
4
[ 2025-04-13 21:49:38 ] Producer Pid : 90132, produce paper. paper_cnt : 4
[ 2025-04-13 21:49:38 ] Consumer Pid : 90136, consume tobacco. got data : 2
[ 2025-04-13 21:49:38 ] Consumer Pid : 90136, consume paper. got data : 3
[ 2025-04-13 21:49:38 ] Consumer Pid : 90135, consume water. got data : 3
[ 2025-04-13 21:49:38 ] Consumer Pid : 90135, consume tobacco. got data : 3
[ 2025-04-13 21:49:39 ] Producer Pid : 90133, produce paper. paper_cnt : 5
```

```
[ 2025-04-13 21:49:39 ] Producer Pid : 90133, produce water. water_cnt : 5
[ 2025-04-13 21:49:39 ] Producer Pid : 90133, produce water. water_cnt : 6
[ 2025-04-13 21:49:39 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
5
[ 2025-04-13 21:49:39 ] Consumer Pid : 90134, consume paper. got data : 4
[ 2025-04-13 21:49:39 ] Consumer Pid : 90134, consume water. got data : 5
[ 2025-04-13 21:49:39 ] Consumer Pid : 90136, consume tobacco. got data : 4
[ 2025-04-13 21:49:39 ] Consumer Pid : 90136, consume paper. got data : 4
[ 2025-04-13 21:49:39 ] Consumer Pid : 90135, consume water. got data : 5
[ 2025-04-13 21:49:39 ] Consumer Pid : 90135, consume tobacco. got data : 5
[ 2025-04-13 21:49:39 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
6
[ 2025-04-13 21:49:40 ] Producer Pid : 90133, produce paper. paper_cnt : 6
[ 2025-04-13 21:49:39 ] Producer Pid : 90132, produce paper. paper_cnt : 5
[ 2025-04-13 21:49:40 ] Producer Pid : 90132, produce water. water_cnt : 5
[ 2025-04-13 21:49:40 ] Producer Pid : 90132, produce water. water_cnt : 6
[ 2025-04-13 21:49:40 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
5
[ 2025-04-13 21:49:40 ] Consumer Pid : 90134, consume paper. got data : 5
[ 2025-04-13 21:49:40 ] Consumer Pid : 90134, consume water. got data : 5
[ 2025-04-13 21:49:40 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
6
[ 2025-04-13 21:49:40 ] Producer Pid : 90132, produce paper. paper_cnt : 6
[ 2025-04-13 21:49:40 ] Consumer Pid : 90135, consume water. got data : 5
[ 2025-04-13 21:49:40 ] Consumer Pid : 90135, consume tobacco. got data : 5
[ 2025-04-13 21:49:40 ] Consumer Pid : 90136, consume tobacco. got data : 4
[ 2025-04-13 21:49:40 ] Consumer Pid : 90136, consume paper. got data : 5
[ 2025-04-13 21:49:41 ] Producer Pid : 90133, produce paper. paper_cnt : 7
[ 2025-04-13 21:49:41 ] Producer Pid : 90133, produce water. water_cnt : 7
[ 2025-04-13 21:49:41 ] Producer Pid : 90133, produce water. water_cnt : 8
[ 2025-04-13 21:49:41 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
7
[ 2025-04-13 21:49:41 ] Consumer Pid : 90134, consume paper. got data : 6
[ 2025-04-13 21:49:41 ] Consumer Pid : 90134, consume water. got data : 7
[ 2025-04-13 21:49:41 ] Consumer Pid : 90136, consume tobacco. got data : 6
[ 2025-04-13 21:49:41 ] Consumer Pid : 90136, consume paper. got data : 6
[ 2025-04-13 21:49:41 ] Consumer Pid : 90135, consume water. got data : 7
[ 2025-04-13 21:49:41 ] Consumer Pid : 90135, consume tobacco. got data : 7
[ 2025-04-13 21:49:41 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
8
[ 2025-04-13 21:49:42 ] Producer Pid : 90133, produce paper. paper_cnt : 8
[ 2025-04-13 21:49:41 ] Producer Pid : 90132, produce paper. paper_cnt : 7
[ 2025-04-13 21:49:42 ] Producer Pid : 90132, produce water. water_cnt : 7
[ 2025-04-13 21:49:42 ] Producer Pid : 90132, produce water. water_cnt : 8
```

```
[ 2025-04-13 21:49:42 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
7
[ 2025-04-13 21:49:42 ] Consumer Pid : 90134, consume paper. got data : 7
[ 2025-04-13 21:49:42 ] Consumer Pid : 90134, consume water. got data : 7
[ 2025-04-13 21:49:42 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
8
[ 2025-04-13 21:49:42 ] Producer Pid : 90132, produce paper. paper_cnt : 8
[ 2025-04-13 21:49:42 ] Consumer Pid : 90135, consume water. got data : 7
[ 2025-04-13 21:49:42 ] Consumer Pid : 90135, consume tobacco. got data : 7
[ 2025-04-13 21:49:42 ] Consumer Pid : 90136, consume tobacco. got data : 6
[ 2025-04-13 21:49:42 ] Consumer Pid : 90136, consume paper. got data : 7
[ 2025-04-13 21:49:43 ] Producer Pid : 90133, produce paper. paper_cnt : 9
[ 2025-04-13 21:49:43 ] Producer Pid : 90133, produce water. water_cnt : 9
[ 2025-04-13 21:49:43 ] Producer Pid : 90133, produce water. water_cnt : 10
[ 2025-04-13 21:49:43 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
9
[ 2025-04-13 21:49:43 ] Consumer Pid : 90134, consume paper. got data : 8
[ 2025-04-13 21:49:43 ] Consumer Pid : 90134, consume water. got data : 9
[ 2025-04-13 21:49:43 ] Consumer Pid : 90135, consume water. got data : 8
[ 2025-04-13 21:49:43 ] Consumer Pid : 90135, consume tobacco. got data : 9
[ 2025-04-13 21:49:43 ] Consumer Pid : 90136, consume tobacco. got data : 9
[ 2025-04-13 21:49:43 ] Consumer Pid : 90136, consume paper. got data : 8
[ 2025-04-13 21:49:43 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
10
[ 2025-04-13 21:49:44 ] Producer Pid : 90133, produce paper. paper_cnt : 10
[ 2025-04-13 21:49:43 ] Producer Pid : 90132, produce paper. paper_cnt : 9
[ 2025-04-13 21:49:44 ] Producer Pid : 90132, produce water. water_cnt : 9
[ 2025-04-13 21:49:44 ] Producer Pid : 90132, produce water. water_cnt : 10
[ 2025-04-13 21:49:44 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
9
[ 2025-04-13 21:49:44 ] Consumer Pid : 90134, consume paper. got data : 9
[ 2025-04-13 21:49:44 ] Consumer Pid : 90134, consume water. got data : 9
[ 2025-04-13 21:49:44 ] Consumer Pid : 90135, consume water. got data : 8
[ 2025-04-13 21:49:44 ] Consumer Pid : 90135, consume tobacco. got data : 9
[ 2025-04-13 21:49:44 ] Consumer Pid : 90136, consume tobacco. got data : 9
[ 2025-04-13 21:49:44 ] Consumer Pid : 90136, consume paper. got data : 9
[ 2025-04-13 21:49:44 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
10
[ 2025-04-13 21:49:44 ] Producer Pid : 90132, produce paper. paper_cnt : 10
[ 2025-04-13 21:49:45 ] Producer Pid : 90133, produce paper. paper_cnt : 11
[ 2025-04-13 21:49:45 ] Producer Pid : 90133, produce water. water_cnt : 11
[ 2025-04-13 21:49:45 ] Producer Pid : 90133, produce water. water_cnt : 12
[ 2025-04-13 21:49:45 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
11
```

```
[ 2025-04-13 21:49:45 ] Consumer Pid : 90134, consume paper. got data : 10
[ 2025-04-13 21:49:45 ] Consumer Pid : 90134, consume water. got data : 11
[ 2025-04-13 21:49:45 ] Consumer Pid : 90135, consume water. got data : 10
[ 2025-04-13 21:49:45 ] Consumer Pid : 90135, consume tobacco. got data : 11
[ 2025-04-13 21:49:45 ] Consumer Pid : 90136, consume tobacco. got data : 11
[ 2025-04-13 21:49:45 ] Consumer Pid : 90136, consume paper. got data : 10
[ 2025-04-13 21:49:45 ] Producer Pid : 90132, produce paper. paper_cnt : 11
[ 2025-04-13 21:49:46 ] Producer Pid : 90132, produce water. water_cnt : 11
[ 2025-04-13 21:49:45 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
12
[ 2025-04-13 21:49:46 ] Producer Pid : 90133, produce paper. paper_cnt : 12
[ 2025-04-13 21:49:46 ] Consumer Pid : 90134, consume paper. got data : 11
[ 2025-04-13 21:49:46 ] Consumer Pid : 90134, consume water. got data : 11
[ 2025-04-13 21:49:46 ] Producer Pid : 90132, produce water. water_cnt : 12
[ 2025-04-13 21:49:46 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
11
[ 2025-04-13 21:49:46 ] Consumer Pid : 90135, consume water. got data : 10
[ 2025-04-13 21:49:46 ] Consumer Pid : 90135, consume tobacco. got data : 11
[ 2025-04-13 21:49:46 ] Consumer Pid : 90136, consume tobacco. got data : 11
[ 2025-04-13 21:49:46 ] Consumer Pid : 90136, consume paper. got data : 11
[ 2025-04-13 21:49:46 ] Producer Pid : 90132, produce tobacco. tobacco_cnt :
12
[ 2025-04-13 21:49:46 ] Producer Pid : 90132, produce paper. paper_cnt : 12
[ 2025-04-13 21:49:47 ] Producer Pid : 90133, produce paper. paper_cnt : 13
[ 2025-04-13 21:49:47 ] Producer Pid : 90133, produce water. water_cnt : 13
[ 2025-04-13 21:49:47 ] Producer Pid : 90133, produce water. water_cnt : 14
[ 2025-04-13 21:49:47 ] Producer Pid : 90133, produce tobacco. tobacco_cnt :
13
[ 2025-04-13 21:49:47 ] Consumer Pid : 90134, consume paper. got data : 12
[ 2025-04-13 21:49:47 ] Consumer Pid : 90134, consume water. got data : 13
[ 2025-04-13 21:49:47 ] Consumer Pid : 90135, consume water. got data : 13
[ 2025-04-13 21:49:47 ] Consumer Pid : 90135, consume tobacco. got data : 13
[ 2025-04-13 21:49:47 ] Consumer Pid : 90136, consume tobacco. got data : 12
[ 2025-04-13 21:49:47 ] Consumer Pid : 90136, consume paper. got data : 12
```

可以根据前方的时间记录来看出生产者生产情况和消费者消费情况。除了因为进程调度顺序导致的日志记录写入的前后区别带来的观测误差，可以看出结果是符合题目的要求的。

结论分析：

问题 1：真实操作系统中提供的并发进程同步机制是怎样实现和解决同步问题的？它们是怎样应用操作系统教材中讲解的进程同步原理的？

回答 1：真实操作系统实际在做的都是维护一块缓冲区，并且通过系统调用的方式为不同的进程返回缓冲区内的内容或者向缓冲区内写入数据，再通过进程调度设置的方式

解决同步问题。

问题 2：对应教材中信号量的定义，说明信号量机制是怎样完成进程的互斥和同步的？其中信号量的初值和其值的变化的物理意义是什么？

回答 2：信号量机制是一种用于进程同步和互斥的工具，它通过两个基本操作 P（等待）和 V（释放）来控制对共享资源的访问。信号量的物理意义通常表示系统内的资源量。

结论：

主要的体会是，操作系统维护的进程通讯方式的本质都是由系统调用实现的操作系统维护的缓冲区与进程之间的通信，并且通过由操作系统来标记进程运行状态的方式来实现进程的阻塞，即等待通信。另外顺便说一下独立实验主要更改了哪些地方。首先使用 static inline 关键字来定义函数，这样可以将函数直接定义在头文件内，也不怕重定义，而且因为函数是小函数，这种方式可以节省额外的链接步骤。然后 key_t 值我们我们选择使用 ftok 函数来生成唯一键值,这样可以保证不会因为键值而导致 ipc 量分布的很乱，并且使用 shmctl 和 semctl 来进行 ipc 量的清理，而不像示例实验那样用完就在这儿放着不管他。我们还编写了.sh 脚本来便利的完成实验过程，这一部分就不赘述了。