

操作系统 课程实验报告

学号：201800130005	姓名：张畅	班级：2018 级计科三班
实验题目：进程同步实验		
实验学时：2	实验日期：2020. 11. 27	
<p>实验目的：</p> <p>加深对并发协作进程同步与互斥概念的理解，观察和体验并发进程同步与互斥操作的效果，分析与研究经典进程同步与互斥问题的实际解决方案。了解 Linux 系统中 IPC 进程同步工具的用法，练习并发协作进程的同步与互斥操作的编程与调试技术。</p>		
<p>实验过程中遇到和解决的问题：</p> <p>（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）</p> <p>在实验的设计实现初期，错误的将题意理解为了两个生产者各生产一个，对应的消费者进行消费的题意。在这种题意下，将程序的整体执行过程规划为：</p> <pre> graph TD A[生产随机一种] --> B[生产一种材料组合] C[生产随机一种] --> B B --> D[两个生产者阻塞] D --> E[生产] E --> F[唤醒对应的消费者] F --> G[消费] G --> H[唤醒两个生产者] H --> A H --> C I[阻塞状态] --> G </pre> <p>整体的程序设计都存在一定的问题，后来逐渐理解了题目中两个消费者的生产规则和具体的执行步骤。在此基础上重新设计了程序的各种同步互斥标识，较好地实现了进程同步实验的需求。此外本实验考虑到生产者在体系中处于决定性的地位，采取了优先消费者的设计理念。</p>		

实验步骤与内容：

1、学习同步互斥问题中使用到的相关知识

在本实验中使用了同步的相关知识，即 IPC 相关的函数调用和机理，可以总结如下：

系统中的同步工具的调用：

创建一段共享内存

```
int shmget(key_t key, int size, int flags);
```

将一段共享内存附加到调用进程

```
char *shmat(int shmid, char *shmaddr, int flags)
```

将一段共享内存从调用进程中分离

```
int shmdt(char *shmadr);
```

创建信号灯数组

```
int semget(key_t key, int nsems, int flags);
```

操作信号灯组

```
int semop(int semid, struct sembuf *semop, unsigned nops);
```

创建消息队列

```
int msgget(key_t key, int flags)
```

追加一条新消息到消息队列

```
int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg);
```

从消息队列读出一条信息

msgrcv 能够实现从消息队列取出**指定类型(msgtype)**的**第一条消息**，并且删除这条消息。

删除消息队列

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

2、阅读并运行测试示例程序，通过运行调试实践 IPC 相关的函数调用

本实验的核心点是利用进程间的同步，实现多个程序进程之间的数据传输和同步，应当保证各个进程访问到的数据是同步一致的。为了实现同步功能，需要掌握三种数据结构的使用方法：共享内存、信号量和消息队列。C 语言为我们提供了多个系统接口方便我们使用三个功能；但是由于原生的函数方法并不够简洁有效，IPC 中提供了一些更加便利的控制方法，能够简单地控制数据从而实现同步和互斥的功能。

经过对给出的实例程序的运行调试，基本掌握了三种数据结构的调试方法：使用共享内存存储和传输数据，使用消息队列在进程之间传递信号，使用信号灯控制同步与互斥的相关功能。结合这三种数据结构，即可完成对本实验的设计。

3、设计实验四的程序逻辑

实验四的功能可以分成两个方面：生产者和消费者。

整体的逻辑可以概括为：生产者生产一种组合的材料，相对应的一个消费者消费并且唤醒生产者继续生产。为了更加清楚的表达这种逻辑，使用图示：

实验总结：

1、并发进程同步机制的作用原理：

进程间的两种关系：同步和互斥。所谓同步就是把异步环境下的一组并发进程，因直接制约而互相发送消息或进行互相合作、互相等待，使得各进程按一定的速度执行的过程；互斥是指不允许两个以上的共享该资源的并发进程同时进入临界区，是进程之间发生的一种间接性作用；

2、信号量（程序中的信号灯）的使用方法

信号量是一种软件资源，是最早出现的用来解决进程同步与互斥问题的机制，包括一个称为信号量的变量及对它进行的两个原语操作，每个信号量至少须记录两个信息：信号量的值和等待该信号量的进程队列。

信号量机制的原理：在操作系统中，信号量 sem 是一整数。在 sem 大于等于零时代表可供并发进程使用的资源实体数，但 sem 小于零时则表示正在等待使用临界区的进程数。显然，用于互斥的信号量 sem 初值应该大于零。信号量数值仅能由 P, V 原语操作改变。执行一次 P 操作意味着请求分配一个单位资源，因此 S 的值减 1；当 $S < 0$ 时，表示已经没有可用资源，请求者必须等待别的进程释放该类资源，它才能运行下去。而执行一个 V 操作意味着释放一个单位资源，因此 S 的值加 1；若 $S = 0$ ，表示有某些进程正在等待该资源，因此要唤醒一个等待状态的进程，使之运行下去。

附录：程序源代码

Producer 程序

```
#include "ipc.h"
```

```
int main(int argc, char *argv[]) {
    int rate;

    //可在在命令行第一参数指定一个进程睡眠秒数，以调解进程执行速度

    if (argv[1] != NULL) rate = atoi(argv[1]);
    else rate = 3; //不指定则默认为 3 秒

    //共享内存使用的变量

    buff_key = 101; //缓冲区任给的键值

    buff_num = 1; //缓冲区任给的长度

    pput_key = 102; //生产者放产品指针的键值

    pput_num = 1; //指针数

    shm_flg = IPC_CREAT | 0644; //共享内存读写权限

    //获取缓冲区使用的共享内存，buff_ptr 指向缓冲区首地址

    buff_ptr = (char *)set_shm(buff_key, buff_num, shm_flg);
```

```

//获取生产者放产品位置指针 pput_ptr

pput_ptr = (int *)set_shm(pput_key, pput_num, shm_flg);

//信号量使用的变量

prod_key = 201;//生产者同步信号灯键值

pmtx_key = 202;//生产者互斥信号灯键值

cons1_key = 301;//消费者 1 同步信号灯键值
cons2_key = 302;//消费者 2 同步信号灯键值
cons3_key = 303;//消费者 3 同步信号灯键值
sem_flg = IPC_CREAT | 0644;

//生产者同步信号灯初值设为缓冲区的容量，这里设置为 1 即可

sem_val = buff_num;
//prod_sem----获取生产者同步信号灯
prod_sem = set_sem(prod_key, sem_val, sem_flg);

//生产者互斥信号灯初值为 1

sem_val = 1;
//pmtx_sem----获取生产者互斥信号灯
pmtx_sem = set_sem(pmtx_key, sem_val, sem_flg);

//由于开始状态缓冲区无存量，三个消费者的同步信号灯初值都设为 0
sem_val = 0;
//获取三个消费者的同步信号 cons1,cons2,cons3
cons1_sem = set_sem(cons1_key, sem_val, sem_flg);
cons2_sem = set_sem(cons2_key, sem_val, sem_flg);
cons3_sem = set_sem(cons3_key, sem_val, sem_flg);

while(1)
{
    //如果缓冲区满则生产者阻塞----生产者的同步信号 prod_sem
    down(prod_sem);
    //如果另一生产者正在放产品，本生产者阻塞---生产者的互斥信号 pmtx_sem
    down(pmtx_sem);

    buff_ptr[*pput_ptr] = 'A' + rand()%3;

```

```

        if(buff_ptr[*pput_ptr] == 'A'){
            printf("The producer(%d) gives tobacco and paper\n",getpid());
            //唤醒阻塞的生产者----生产者的互斥信号 pmtx_sem
            up(pmtx_sem);
            //唤醒一号消费者----消费者 1 的同步信号 cons1_sem
            up(cons1_sem);
        }
        if(buff_ptr[*pput_ptr] == 'B'){
            printf("The producer(%d) gives tobacco and glue\n",getpid());
            //唤醒阻塞的生产者----生产者的互斥信号 pmtx_sem
            up(pmtx_sem);
            //唤醒二号消费者----消费者 2 的同步信号 cons2_sem
            up(cons2_sem);
        }
        if(buff_ptr[*pput_ptr] == 'C'){
            printf("The producer(%d) gives glue and paper\n",getpid());
            //唤醒阻塞的生产者----生产者的互斥信号 pmtx_sem
            up(pmtx_sem);
            //唤醒三号消费者----消费者 3 的同步信号 cons3_sem
            up(cons3_sem);
        }
        sleep(rate);
    }
    return EXIT_SUCCESS;
}

```

Consumer 程序:

```
#include "ipc.h"
```

```

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

    //可在在命令行第一参数指定一个进程睡眠秒数，以调解进程执行速度
    if (argv[1] != NULL) rate = atoi(argv[1]);
    else rate = 3; //不指定为 3 秒

    //共享内存 使用的变量

    buff_key = 101; //缓冲区任给的键值

    buff_num = 1; //缓冲区任给的长度

    cget_key = 103; //消费者取产品指针的键值
    cget_num = 1; //指针数
    shm_flg = IPC_CREAT | 0644; //共享内存读写权限
}

```

```

//获取缓冲区使用的共享内存，buff_ptr 指向缓冲区首地址
buff_ptr = (char *)set_shm(buff_key, buff_num, shm_flg);

//获取消费者取产品指针，cget_ptr 指向索引地址
cget_ptr = (int *)set_shm(cget_key, cget_num, shm_flg);

//信号量使用的变量

prod_key = 201; //生产者同步信号灯键值

cons1_key = 301; //消费者同步信号灯键值
sem_flg = IPC_CREAT | 0644; //信号灯操作权限

//生产者同步信号灯初值设为缓冲区大小，这里设置为 1 即可
sem_val = buff_num;

//prod_sem----生产者同步信号灯
prod_sem = set_sem(prod_key, sem_val, sem_flg);

//消费者初始无产品可取，同步信号灯初值设为 0
sem_val = 0;
//cons1_sem----消费者 1 同步信号灯
cons1_sem = set_sem(cons1_key, sem_val, sem_flg);

//循环执行模拟消费者不断取产品
while (1) {
    //如果自己的信号灯为 0，则阻塞
    down(cons1_sem);

    //模拟消费者取产品，报告本进程的进程号和获取的字符
    sleep(rate);
    printf("%d consumer(have glue):now get tobacco and paper\n", getpid());

    //唤醒阻塞的生产者
    up(prod_sem);
}
return EXIT_SUCCESS;
}

```