**信号量实现进程同步**

**实验内容：**

生产者进程生产产品，消费者进程消费产品。当生产者进程生产产品时，如果没有空缓冲区（仓库）可用，那么生产进程必须等待消费者进程释放出一个缓冲区，当消费者进程消费产品时，如果缓冲区产品，那么消费者进程将被阻塞，直到新的产品被生产出来。模拟一个生产者两个消费者的情况。

**实验分析：**

定义3个信号量，full，empty，mutex，分别表示产品个数，缓冲区空位个数，对缓冲区进行操作的互斥信号量，对应的初始化值分别为0，n，1。

生产者：

P（empty）---->P（mutex）----->V（mutex）----->V（full）

消费者：

P（full）----->P（mutex）------->V（mutex）------->V（empty）

**函数：**

可参考：http://blog.csdn.net/ljianhui/article/details/10243617

1 semget函数

|  |
| --- |
| 它的作用是创建一个新信号量或取得一个已有信号量，原型为：  int semget(key\_t  key, int num\_sems, int sem\_flags);  semget函数成功返回一个相应信号标识符（非零），失败返回-1.  第一个参数key是整数值（唯一非零），不相关的进程可以通过它访问一个信号量，它代表程序可能要使用的某个资源，程序对所有信号量的访问都是间接的，程序先通过调用semget函数并提供一个键，再由系统生成一个相应的信号标识符（semget函数的返回值），只有semget函数才直接使用信号量键，所有其他的信号量函数使用由semget函数返回的信号量标识符。如果多个程序使用相同的key值，key将负责协调工作。  第二个参数num\_sems指定需要的信号量数目，它的值几乎总是1。  第三个参数sem\_flags是一组标志，当想要当信号量不存在时创建一个新的信号量，可以和值IPC\_CREAT做按位或操作。设置了IPC\_CREAT标志后，即使给出的键是一个已有信号量的键，也不会产生错误。而IPC\_CREAT | IPC\_EXCL则可以创建一个新的，唯一的信号量，如果信号量已存在，返回一个错误。 |

2 semop函数

|  |
| --- |
| 它的作用是改变信号量的值，原型为：  int semop(int sem\_id, struct sembuf \*sem\_opa, size\_tum\_sem\_ops);  sem\_id是由semget返回的信号量标识符，sembuf结构的定义如下：  struct sembuf{      short sem\_num;//除非使用一组信号量，否则它为0      short sem\_op;//信号量在一次操作中需要改变的数据，通常是两个数，一个是-1，P（等待）操作，一个是+1，即V（发送信号）操作。       short sem\_flg;//通常为SEM\_UNDO,使操作系统跟踪信号，并在进程没有释放该信号量而终止时，操作系统释放信号量  }; |

3 semctl函数

|  |
| --- |
| 该函数用来直接控制信号量信息，它的原型为：  int semctl(int sem\_id, int sem\_num, int command, ...);  如果有第四个参数，它通常是一个union semum结构，定义如下：  union semun{      int val;      struct semid\_ds \*buf;      unsigned short \*arry;  };  前两个参数与前面一个函数中的一样，command通常是下面两个值中的其中一个  SETVAL：用来把信号量初始化为一个已知的值。这个值通过union semun中的val成员设置，其作用是在信号量第一次使用前对它进行设置。  IPC\_RMID：用于删除一个已经无需继续使用的信号量标识符。 |

**程序实现**

定义PV操作函数。例如P操作可以定义如下：

|  |
| --- |
| Int semaphore\_p(int sem\_id)  {  //对信号量做减1操作，即等待P（sv）  struct sembuf sem\_b;  sem\_b.sem\_num = 0;  sem\_b.sem\_op = -1;//P()  sem\_b.sem\_flg = SEM\_UNDO;  if(semop(sem\_id, &sem\_b, 1) == -1)  {  fprintf(stderr, "semaphore\_p failed\n");  return 0;  }  return 1;  } |

主函数如下：

|  |
| --- |
| int main(int argc, char \*argv[])  {  //创建信号量,利用semget函数  //初始化信号量，利用semctl函数  pid\_t p1,p2;  if((p1=fork())==0){  while(1){  生产者……………..  sleep(2);  }  }else{  if((p2=fork())==0){  while(1){  sleep(2);  消费者……………..  sleep(5);  }  }else{  while(1){  sleep(2);  消费者…………….  sleep(5);  }  }  }  } |

代码主要框架：

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

#include<sys/sem.h>

union semun{

    int val;

    struct semid\_ds \*buf;

    unsigned short \*arry;

};

//基本量定义（三种信号量、P操作、V操作、初始化/删除信号量）

int main(intargc, char \*argv[])

{

//创建信号量

//程序第一次被调用，初始化信号量

//set\_semvalue三种信号量的判断

pid\_t p1,p2;

//pv操作

}

staticint set\_semvalue(intsem\_id,intvalue)

{

//用于初始化信号量，在使用信号量前必须这样做

}

staticvoid del\_semvalue(intsem\_id)

{

//删除信号量

}

staticint semaphore\_p(intsem\_id)

{

//对信号量做减1操作，即等待P（sv）

}

staticint semaphore\_v(intsem\_id)

{

//一个释放操作，它使信号量变为可用，即发送信号V（sv）

}