**大实验-信号量实现**

### 实验目的

* 加深对进程同步与互斥概念的认识；
* 掌握信号量的使用，并应用它解决生产者——消费者问题；
* 掌握信号量的实现原理。

### 实验内容

* 在 Ubuntu 下编写程序，用信号量解决生产者——消费者问题；
* 在 0.11 中实现信号量，用生产者—消费者程序检验之。

#### 信号量系统调用

sem\_t\* sys\_sem\_open(const char\* name, unsigned int value)  
  
int sys\_sem\_wait(sem\_t \*sem)  
  
int sys\_sem\_post(sem\_t \*sem)  
  
int sys\_sem\_unlink(const char\* name)

C

* sem\_open() 的功能是创建一个信号量，或打开一个已经存在的信号量。
  + sem\_t 是信号量类型，根据实现的需要自定义。
  + name 是信号量的名字。不同的进程可以通过提供同样的 name 而共享同一个信号量。如果该信号量不存在，就创建新的名为 name 的信号量；如果存在，就打开已经存在的名为 name 的信号量。
  + value 是信号量的初值，仅当新建信号量时，此参数才有效，其余情况下它被忽略。当成功时，返回值是该信号量的唯一标识（比如，在内核的地址、ID 等），由另两个系统调用使用。如失败，返回值是 NULL。
* sem\_wait() 就是信号量的 P 原子操作。如果继续运行的条件不满足，则令调用进程等待在信号量 sem 上。返回 0 表示成功，返回 -1 表示失败。
* sem\_post() 就是信号量的 V 原子操作。如果有等待 sem 的进程，它会唤醒其中的一个。返回 0 表示成功，返回 -1 表示失败。
* sem\_unlink() 的功能是删除名为 name 的信号量。返回 0 表示成功，返回 -1 表示失败。

#### 生产者消费者问题

利用上面实现的信号量系统调用，编写一个应用程序“pc.c”来模拟经典的生产者 消费者之间的同步。

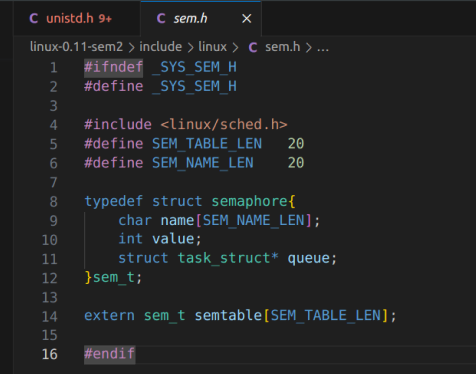
在这个程序中，要建立 1 个生产者进程，5 个消费者进程；用文件建立一个共享缓冲区；生产者进程依次向缓冲区写入整数 0,1,2,...,499；每个消费者进程从缓冲区中读取 100 个数，每读取一个数字就打印到标准输出上；缓冲区文件最多只能保存 10 个数。

### 实验步骤

#### 1 添加信号量系统调用

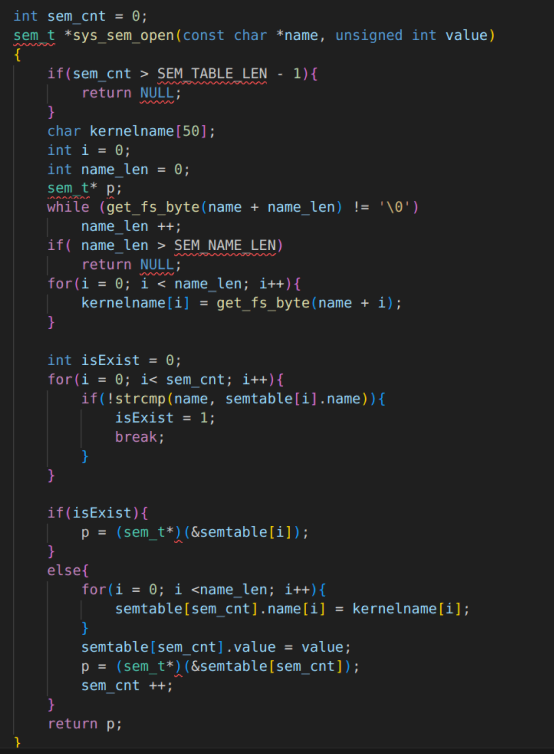
1. 修改unist.h与sys.h文件，添加有关信号量系统调用的信息。
2. 添加信号量头文件sem.h与信号量系统调用实现文件sem.c。

头文件中声明了信号量的结构体，包括name, value, queue三个成员。



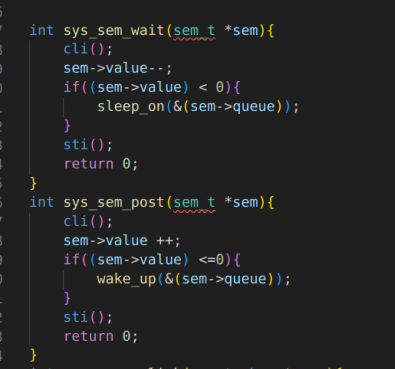
信号量系统调用实现：

sys\_sem\_open

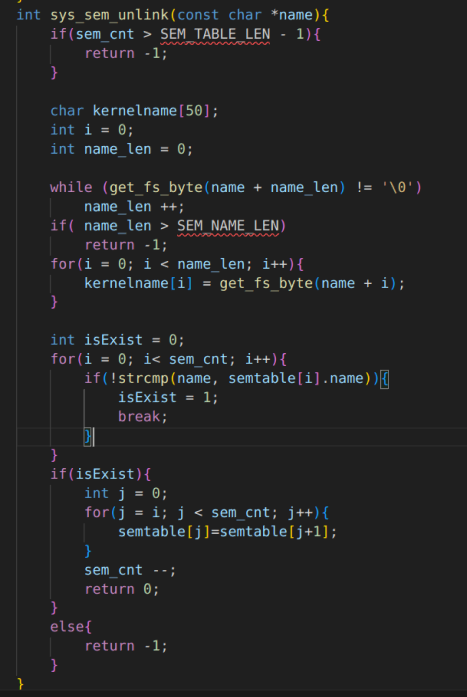


sys\_sem\_wait与sys\_sem\_post

通过关闭中断保护临界区



sys\_sem\_unlink



#### 2 编写生产者-消费者程序

生产者-父进程

for (item\_pro = 0; item\_pro < ITEM\_COUNT; item\_pro++)  
 {  
 sem\_wait(empty);  
 sem\_wait(mutex);  
 /\* 读写文件 \*/  
 if (!(item\_pro % BUFFER\_SIZE)){  
 printf("pid: %d \t, item\_pro: %d \t write at the end\n", pid, item\_pro);  
 lseek(fw, 0, 0);  
 }  
 write(fw, (char \*)&item\_pro, sizeof(int));  
  
 printf("pid %d:\tproduces item %d\n", pid, item\_pro);  
 fflush(stdout);  
 sem\_post(mutex);  
 sem\_post(full);  
 }

C

消费者-子进程

for (i = 0; i < CONSUMER\_COUNT; i++)  
 {  
 pid = fork();  
  
 if (pid == 0)  
 {  
 pid = getpid();  
 printf("consumer pid: %d\n", pid);  
 fflush(stdout);  
  
 while (1)  
 {  
 sem\_wait(full);  
 sem\_wait(mutex);  
 /\* 读写文件 \*/  
  
 /\* read(fr, (char \*)&item\_used, sizeof(int)); \*/  
 if (!read(fr, (char \*)&item\_used, sizeof(item\_used)))  
 {  
 printf("pid: %d \t read at the end\n", pid);  
 lseek(fr, 0, 0);  
 read(fr, (char \*)&item\_used, sizeof(item\_used));  
 }  
 printf("pid %d:\t consumes item %d\n", pid, item\_used);  
 fflush(stdout);  
  
 sem\_post(empty); /\* 唤醒生产者进程 \*/  
 sem\_post(mutex);  
  
 if (item\_used == ITEM\_COUNT)  
 goto OK;  
 }  
 }  
 }

C

#### 3 测试

