**信号量的实现和应用**

目录

1. [实验目的](https://cms.hit.edu.cn/mod/wiki/view.php?pageid=26#toc-1)[[编辑]](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E5%AE%9E%E9%AA%8C%E7%9B%AE%E7%9A%84)

2. [实验内容](https://cms.hit.edu.cn/mod/wiki/view.php?pageid=26#toc-2)[[编辑]](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E5%AE%9E%E9%AA%8C%E5%86%85%E5%AE%B9)

3. [实验报告](https://cms.hit.edu.cn/mod/wiki/view.php?pageid=26#toc-3)[[编辑]](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E5%AE%9E%E9%AA%8C%E6%8A%A5%E5%91%8A)

4. [评分标准](https://cms.hit.edu.cn/mod/wiki/view.php?pageid=26#toc-4)[[编辑]](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E8%AF%84%E5%88%86%E6%A0%87%E5%87%86)

5. [实验提示](https://cms.hit.edu.cn/mod/wiki/view.php?pageid=26#toc-5)[[编辑]](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E5%AE%9E%E9%AA%8C%E6%8F%90%E7%A4%BA)

难度系数：★★★★☆

**实验目的[[编辑]](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E5%AE%9E%E9%AA%8C%E7%9B%AE%E7%9A%84)**

* 加深对进程[同步](https://cms.hit.edu.cn/mod/quiz/view.php?id=2011" \o "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[[编辑]](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E5%AE%9E%E9%AA%8C%E5%86%85%E5%AE%B9)**

本次实验的基本内容是：

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

**用信号量解决生产者—消费者问题**

在Ubuntu上编写应用程序“pc.c”，解决经典的生产者—消费者问题，完成下面的功能：

1. 建立一个生产者进程，N个消费者进程（N>1）；
2. 用文件建立一个共享缓冲区；
3. 生产者进程依次向缓冲区写入整数0,1,2,...,M，M>=500；
4. 消费者进程从缓冲区读数，每次读一个，并将读出的数字从缓冲区删除，然后将本进程ID和数字输出到标准输出；
5. 缓冲区同时最多只能保存10个数。

一种可能的输出效果是：

10: 0  
10: 1  
10: 2  
10: 3  
10: 4  
11: 5  
11: 6  
12: 7  
10: 8  
12: 9  
12: 10  
12: 11  
12: 12  
……  
11: 498  
11: 499

其中ID的顺序会有较大变化，但冒号后的数字一定是从0开始递增加一的。

pc.c中将会用到sem\_open()、sem\_close()、sem\_wait()和sem\_post()等信号量相关的系统调用，请查阅相关文档。

《UNIX环境高级编程》是一本关于Unix/Linux系统级编程的相当经典的教程。校园网用户可以在 <ftp://run.hit.edu.cn/study/Computer_Science/Linux_Unix/> 下载，后续实验也用得到。如果你对POSIX编程感兴趣，建议买一本常备手边。

**实现信号量**

Linux在0.11版还没有实现信号量，Linus把这件富有挑战的工作留给了你。如果能实现一套山寨版的完全[符合POSIX规范的信号量](http://linux.die.net/man/7/sem_overview" \o "信号量manpage)，无疑是很有成就感的。但时间暂时不允许我们这么做，所以先弄一套缩水版的类POSIX信号量，它的函数原型和标准并不完全相同，而且只包含如下系统调用：

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

sem\_t是信号量类型，根据实现的需要自定义。

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。

* 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。

在kernel目录下新建“sem.c”文件实现如上功能。然后将pc.c从Ubuntu移植到0.11下，测试自己实现的信号量。

**实验报告**[**[编辑]**](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E5%AE%9E%E9%AA%8C%E6%8A%A5%E5%91%8A)

完成实验后，在实验报告中回答如下问题：

1. 在pc.c中去掉所有与信号量有关的代码，再运行程序，执行效果有变化吗？为什么会这样？
2. 实验的设计者在第一次编写生产者——消费者程序的时候，是这么做的：

Producer()  
{  
 P(Mutex); //互斥信号量  
 生产一个产品item;  
 P(Empty); //空闲缓存资源  
 将item放到空闲缓存中;  
 V(Full); //产品资源  
 V(Mutex);  
}  
  
Consumer()  
{  
 P(Mutex);   
 P(Full);   
 从缓存区取出一个赋值给item;  
 V(Empty);  
 消费产品item;  
 V(Mutex);  
}

这样可行吗？如果可行，那么它和标准解法在执行效果上会有什么不同？如果不可行，那么它有什么问题使它不可行？

**评分标准**[**[编辑]**](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E8%AF%84%E5%88%86%E6%A0%87%E5%87%86)

* pc.c，40%
* sem\_open()，10%
* sem\_post()，10%
* sem\_wait()，10%
* sem\_unlink()，10%
* 实验报告，20%

**实验提示[[编辑]](https://cms.hit.edu.cn/mod/wiki/edit.php?pageid=26&section=%E5%AE%9E%E9%AA%8C%E6%8F%90%E7%A4%BA)**

**信号量**

信号量，英文为semaphore，最早由荷兰科学家、图灵奖获得者E. W. Dijkstra设计，任何操作系统教科书的“进程[同步](https://cms.hit.edu.cn/mod/quiz/view.php?id=2011" \o "同步)”部分都会有详细叙述。

Linux的信号量秉承POSIX规范，用“man sem\_overview”可以查看相关信息。本次实验涉及到的信号量系统调用包括：sem\_open()、sem\_wait()、sem\_post()和sem\_unlink()。

**生产者—消费者问题**

生产者—消费者问题的解法几乎在所有操作系统教科书上都有，其基本结构为：

Producer()  
{  
 生产一个产品item;  
 P(Empty); //空闲缓存资源  
 P(Mutex); //互斥信号量  
 将item放到空闲缓存中;  
 V(Mutex);  
 V(Full); //产品资源  
}  
  
Consumer()  
{  
 P(Full);   
 P(Mutex);   
 从缓存区取出一个赋值给item;  
 V(Mutex);  
 V(Empty);  
 消费产品item;  
}

显然在演示这一过程时需要创建两类进程，一类执行函数Producer()，另一类执行函数Consumer()。

**多进程共享文件**

在Linux下使用C语言，可以通过三种方法进行文件的读写：

1. 使用标准C的fopen()、fread()、fwrite()、fseek()和fclose()等；
2. 使用系统调用open()、read()、write()、lseek()和close()等；
3. 通过内存镜像文件，使用mmap()系统调用。

在Linux 0.11上只能使用前两种方法。

fork()调用成功后，子进程会继承父进程拥有的大多数资源，包括父进程打开的文件。所以子进程可以直接使用父进程创建的文件指针/描述符/句柄，访问的是与父进程相同的文件。

使用标准C的文件操作函数要注意，它们使用的是进程空间内的文件缓冲区，父进程和子进程之间不共享这个缓冲区。因此，任何一个进程做完写操作后，必须fflush()一下，将数据强制更新到磁盘，其它进程才能读到所需数据。

建议直接使用系统调用进行文件操作。

**终端也是临界资源**

用printf()向终端输出信息是很自然的事情，但当多个进程同时输出时，终端也成为了一个临界资源，需要做好互斥保护，否则输出的信息可能错乱。

另外，printf()之后，信息只是保存在输出缓冲区内，还没有真正送到终端上，这也可能造成输出信息时序不一致。用fflush(stdout)可以确保数据送到终端。

**原子操作、睡眠和唤醒**

Linux 0.11是一个支持并发的现代操作系统，虽然它还没有面向应用实现任何锁或者信号量，但它内部一定使用了锁机制，即在多个进程访问共享的内核数据时一定需要通过锁来实现互斥和[同步](https://cms.hit.edu.cn/mod/quiz/view.php?id=2011" \o "同步)。锁必然是一种原子操作。通过模仿0.11的锁，就可以实现信号量。

多个进程对磁盘的并发访问是一个需要锁的地方。Linux 0.11访问磁盘的基本处理办法是在内存中划出一段磁盘缓存，用来加快对磁盘的访问。进程提出的磁盘访问请求首先要到磁盘缓存中去找，如果找到直接返回；如果没有找到则申请一段空闲的磁盘缓存，以这段磁盘缓存为参数发起磁盘读写请求。请求发出后，进程要睡眠等待（因为磁盘读写很慢，应该让出CPU让其他进程执行）。这种方法是许多操作系统（包括现代Linux、UNIX等）采用的较通用的方法。这里涉及到多个进程共同操作磁盘缓存，而进程在操作过程可能会被[调度](https://cms.hit.edu.cn/mod/quiz/view.php?id=2010" \o "调度)而失去CPU。因此操作磁盘缓存时需要考虑互斥问题，所以其中必定用到了锁。而且也一定用到了让进程睡眠和唤醒。

下面是从kernel/blk\_drv/ll\_rw\_blk.c文件中取出的两个函数：

static inline void lock\_buffer(struct buffer\_head \* bh)  
{  
 cli(); //关中断  
 while (bh->b\_lock)  
 sleep\_on(&bh->b\_wait); //将当前进程睡眠在bh->b\_wait  
 bh->b\_lock=1;  
 sti(); //开中断  
}  
  
static inline void unlock\_buffer(struct buffer\_head \* bh)  
{  
 if (!bh->b\_lock)  
 printk("ll\_rw\_block.c: buffer not locked\n\r");  
 bh->b\_lock = 0;  
 wake\_up(&bh->b\_wait); //唤醒睡眠在bh->b\_wait上的进程  
}

分析lock\_buffer()可以看出，访问锁变量时用开、关中断来实现原子操作，阻止进程切换的发生。当然这种方法有缺点，且不适合用于多处理器环境中，但对于Linux 0.11，它是一种简单、直接而有效的机制。

另外，上面的函数表明Linux 0.11提供了这样的接口：用sleep\_on()实现进程的睡眠，用wake\_up()实现进程的唤醒。它们的参数都是一个结构体指针——struct task\_struct \*，即进程都睡眠或唤醒在该参数指向的一个进程PCB结构链表上。

因此，我们可以用开关中断的方式实现原子操作，而调用sleep\_on()和wake\_up()进行进程的睡眠和唤醒。

sleep\_on()的功能是将当前进程睡眠在参数指定的链表上（注意，这个链表是一个隐式链表，详见《注释》一书）。wake\_up()的功能是唤醒链表上睡眠的所有进程。这些进程都会被[调度](https://cms.hit.edu.cn/mod/quiz/view.php?id=2010" \o "调度)运行，所以它们被唤醒后，还要重新判断一下是否可以继续运行。可参考lock\_buffer()中的那个while循环。

**应对混乱的bochs虚拟屏幕**

不知是Linux 0.11还是bochs的bug，如果向终端输出的信息较多，bochs的虚拟屏幕会产生混乱。此时按ctrl+L可以重新初始化一下屏幕，但输出信息一多，还是会混乱。建议把输出信息重定向到一个文件，然后用vi、more等工具按屏查看这个文件，可以基本解决此问题。

**string.h**

下面描述的问题未必具有普遍意义，仅做为提醒，请实验者注意。

include/string.h实现了全套的C语言字符串操作，而且都是采用汇编+inline方式优化。但在使用中，某些情况下可能会遇到一些奇怪的问题。比如某人就遇到strcmp()会破坏参数内容的问题。如果调试中遇到有些“诡异”的情况，可以试试不包含头文件，一般都能解决。不包含string.h，就不会用inline方式调用这些函数，它们工作起来就趋于正常了。

如果遇到类似问题，欢迎到论坛说明，进行更深入的分析