阻塞操作是指，在执行设备操作时，若不能获得资源，则进程挂起直到满足可操作的条 件再进行操作。非阻塞操作的进程在不能进行设备操作时，并不挂起。被挂起的进程进入sleep状态，被从调度器的运行队列移走，直到等待的条件被满足。

在Linux驱动程序中，我们可以使用等待队列（wait queue）来实现阻塞操作。

wait queue 很早就作为一个基本的功能单位出现在Linux内核里了，它以队列为基础数据结构，与进程 调度机制紧密结合，能够用于实现核心的异步事件通知机制。等待队列可以用来同步对系统 资源的访问，上节中所讲述Linux信号量在内核中也是由等待队列来实现的。

下面我们重新定义设备“globalvar"，它可以被多个进程打开，但是每次只有当一个进 程写入了一个数据之后本进程或其它进程才可以读取该数据，否则一直阻塞。

#include <linux/module.h>

#include <linux/init.h>

#include <linux/fs.h>

#include <asm/uaccess.h>

#include <linux/wait.h>

#include <asm/semaphore.h>

MODULE\_LICENSE("GPL");

#define MAJOR\_NUM 254

static ssize\_t globalvar\_read(struct file \*, char \*, size\_t, loff\_t\*);

static ssize\_t globalvar\_write(struct file \*, const char \*, size\_t, loff\_t\*);

struct file\_operations globalvar\_fops =

{

read: globalvar\_read,

write: globalvar\_write,

};

static int global\_var = 0;

static struct semaphore sem;

static wait\_queue\_head\_t outq;//定义队列头

static int flag = 0;

static int \_\_init globalvar\_init(void)

{

int ret;

ret = register\_chrdev(MAJOR\_NUM, "globalvar", &globalvar\_fops);

if (ret)

{

printk("globalvar register failure");

}

else

{

printk("globalvar register success");

init\_MUTEX(&sem);

init\_waitqueue\_head(&outq);

}

return ret;

}

static void \_\_exit globalvar\_exit(void)

{

int ret;

ret = unregister\_chrdev(MAJOR\_NUM, "globalvar");

if (ret)

{

printk("globalvar unregister failure");

}

else

{

printk("globalvar unregister success");

}

}

static ssize\_t globalvar\_read(struct file \*filp, char \*buf, size\_t len, loff\_t \*off)

{

    //等待数据可获得，等待一个事件发生，需要唤醒加条件满足

if (wait\_event\_interruptible(outq, flag != 0))

    {

        return -ERESTARTSYS;//因为信号打断而返回

}

if (down\_interruptible(&sem))

{

return -ERESTARTSYS;

}

flag = 0;

if (copy\_to\_user(buf, &global\_var, sizeof(int)))

{

up(&sem);

return -EFAULT;

}

up(&sem);

return sizeof(int);

}

static ssize\_t globalvar\_write(struct file \*filp, const char \*buf, size\_t len,loff\_t \*off)

{

if (down\_interruptible(&sem))

{

return -ERESTARTSYS;

}

if (copy\_from\_user(&global\_var, buf, sizeof(int)))

{

up(&sem);

return -EFAULT;

}

up(&sem);

         flag = 1;

         //唤醒

wake\_up\_interruptible(&outq);

return sizeof(int);

}

module\_init(globalvar\_init);

module\_exit(globalvar\_exit);

编写两个用户态的程序来测试，第一个用于阻塞地读/dev/globalvar，另一个用于写/dev/globalvar。只有当后一个对/dev/globalva进行了输入之后，前者的read才能返回。

读的程序为：

#include <sys/types.h>

#include <sys/stat.h>

#include <stdio.h>

#include <fcntl.h>

main()

{

int fd, num;

fd = open("/dev/globalvar", O\_RDWR, S\_IRUSR | S\_IWUSR);

if (fd != - 1)

{

while (1)

{

read(fd, &num, sizeof(int)); //程序将阻塞在此语句，除非有针对 globalvar的输入

printf("The globalvar is %d\n", num);

//如果输入是0，则退出

if (num == 0)

{

close(fd);

break;

}

}

}

else

{

printf("device open failure\n");

}

}

写的程序为：

#include <sys/types.h>

#include <sys/stat.h>

#include <stdio.h>

#include <fcntl.h>

main()

{

int fd, num;

fd = open("/dev/globalvar", O\_RDWR, S\_IRUSR | S\_IWUSR);

if (fd != - 1)

{

while (1)

{

printf("Please input the globalvar:\n");

scanf("%d", &num);

write(fd, &num, sizeof(int));

//如果输入0，退出

if (num == 0)

{

close(fd);

break;

}

}

}

else

{

printf("device open failure\n");

}

}

打开两个终端，分别运行上述两个应用程序，发现当在第二个终端中没有输入数据时，第一个终端没有输出（阻塞），每当我们在第二个终端中给globalvar输入一个值，第一个终端就会输出这个值

如果将驱动程序中的read函数改为：

static ssize\_t globalvar\_read(struct file \*filp, char \*buf, size\_t len, loff\_t \*off)

{

//获取信号量：可能阻塞

if (down\_interruptible(&sem))

{

return -ERESTARTSYS;

}

//等待数据可获得：可能阻塞

if (wait\_event\_interruptible(outq, flag != 0))

{

return -ERESTARTSYS;

}

flag = 0;

//临界资源访问

if (copy\_to\_user(buf, &global\_var, sizeof(int)))

{

up(&sem);

return -EFAULT;

}

//释放信号量

up(&sem);

return sizeof(int);

}

即交换wait\_event\_interruptible(outq, flag != 0)和down\_interruptible(&sem)的顺序，这个 驱动程序将变得不可运行。实际上，当两个可能要阻塞的事件同时出现时，即两个wait\_event 或down摆在一起的时候，将变得非常危险，死锁的可能性很大，这个时候我们要特别留意

它们的出现顺序。当然，我们应该尽可能地避免这种情况的发生！

+还有一个与设备阻塞与非阻塞访问息息相关的论题，即

select和poll，select和poll的 本质一样，前者在BSD Unix中引入，后者在System V中引入。

poll和select用于查询设备 的状态，以便用户程序获知是否能对设备进行非阻塞的访问，它们都需要设备驱动程序中的poll函数支持。

驱动程序中poll函数中最主要用到的一个API是poll\_wait，其原型如下：

void poll\_wait(struct file \*filp, wait\_queue\_heat\_t \*queue, poll\_table \* wait);

poll\_wait函数所做的工作是把当前进程添加到wait参数指定的等待列表(poll\_table)中。下面我们给globalvar的驱动添加一个poll函数：

static unsigned int globalvar\_poll(struct file \*filp, poll\_table \*wait)

{

unsigned int mask = 0;

poll\_wait(filp, &outq, wait);

//数据是否可获得？

if (flag != 0)

{

mask |= POLLIN | POLLRDNORM; //标示数据可获得

 }

 return mask;

}

需要说明的是，poll\_wait函数并不阻塞，程序中poll\_wait(filp, &outq, wait)这句话的意 思并不是说一直等待

outq信号量可获得，真正的阻塞动作是上层的select/poll函数中完成的。

select/poll会在一个循环中对每个需要监听的设备调用它们自己的poll支持函数以使得当前 进程被加入各个设备的等待列表。若当前没有任何被监听的设备就绪，则内核进行调度（调 用schedule）让出cpu进入阻塞状态，schedule返回时将再次循环检测是否有操作可以进行，如此反复；否则，若有任意一个设备就绪，select/poll都立即返回。

我们编写一个用户态应用程序来测试改写后的驱动。程序中要用到BSD Unix中引入的select函数，其原型为：

int select(int numfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout);

其中 readfds、writefds、exceptfds分别是被select()监视的读、写和异常处理的文件描述 符集合，numfds的值是需要检查的号码最高的文件描述符加 1。timeout参数是一个指向struct timeval类型的指针，它可以使select()在等待timeout时间后若没有文件描述符准备好则返回。

struct timeval数据结构为：

struct timeval

{

int tv\_sec; /\* seconds \*/

int tv\_usec; /\* microseconds \*/

};

除此之外，我们还将使用下列 API：

FD\_ZERO(fd\_set \*set)――清除一个文件描述符集；

FD\_SET(int fd,fd\_set \*set)――将一个文件描述符加入文件描述符集中；

FD\_CLR(int fd,fd\_set \*set)――将一个文件描述符从文件描述符集中清除；

FD\_ISSET(int fd,fd\_set \*set)――判断文件描述符是否被置位。

下面的用户态测试程序等待/dev/globalvar可读，但是设置了 5秒的等待超时，若超过 5

秒仍然没有数据可读，则输出“No data within 5 seconds”：

#include <sys/types.h>

#include <sys/stat.h>

#include <stdio.h>

#include <fcntl.h>

#include <sys/time.h>

#include <sys/types.h>

#include <unistd.h>

main()

{

int fd, num;

fd\_set rfds;

struct timeval tv;

fd = open("/dev/globalvar", O\_RDWR, S\_IRUSR | S\_IWUSR);

if (fd != - 1)

{

while (1)

{

//查看 globalvar是否有输入

FD\_ZERO(&rfds);

FD\_SET(fd, &rfds);

//设置超时时间为 5s

tv.tv\_sec = 5;

tv.tv\_usec = 0;

select(fd + 1, &rfds, NULL, NULL, &tv);

//数据是否可获得？

if (FD\_ISSET(fd, &rfds))

{

read(fd, &num, sizeof(int));

printf("The globalvar is %d\n", num);

//输入为 0，退出

if (num == 0)

{

close(fd);

break;

}

}

else

{

printf("No data within 5 seconds.\n");

}

}

}

else

{

printf("device open failure\n");

}

}

 开两个终端，分别运行程序：一个对globalvar进行写，一个用上述程序对globalvar进行读。当我们在写终端给globalvar输入一个值后，读终端立即就能输出该值，当我们连续5 秒没有输入时，“No data within 5 seconds”在读终端被输出，