
阅码场
YOMOCODE

Linux进程、线程和调度(6)

麦当劳喜欢您来，喜欢您再来

第六次课大纲

- 深度睡眠TASK_UNINTERRUPTIBLE和浅度睡眠TASK_INTERRUPTIBLE
- D状态的意思是什么？
- 为什么需要深度睡眠？
- TASK_KILLABLE

练习题

- 写内核代码，实践TASK_UNINTERRUPTIBLE、TASK_INTERRUPTIBLE和TASK_KILLABLE

睡眠代码模板

```
static ssize_t globalfifo_read(struct file *filp, char __user *buf,
                               size_t count, loff_t *ppos)
{
        int ret;
        struct globalfifo_dev *dev = container_of(filp->private_data,
                struct globalfifo_dev, miscdev);

        DECLARE_WAITQUEUE(wait, current);

        mutex_lock(&dev->mutex);
        add_wait_queue(&dev->r_wait, &wait);

        while (dev->current_len == 0) {
                if (filp->f_flags & O_NONBLOCK) {
                        ret = -EAGAIN;
                        goto out;
                }
                __set_current_state(TASK_INTERRUPTIBLE);
                mutex_unlock(&dev->mutex);

                schedule();
                if (signal_pending(current)) {
                        ret = -ERESTARTSYS;
                        goto out2;
                }

                mutex_lock(&dev->mutex);
        }

        if (count > dev->current_len)
                count = dev->current_len;

        if (copy_to_user(buf, dev->mem, count)) {
                ret = -EFAULT;
                goto out;
        } else {
                memcpy(dev->mem, dev->mem + count, dev->current_len - count);
                dev->current_len -= count;
                printk(KERN_INFO "read %d bytes(s),current_len:%d\n", count,
```

深度睡眠不可避免

there are many loops that cannot exit
until they have a result, simply because the calling conventions require that. The most common example is disk wait.

HOWEVER, if they are killed by a signal, the calling convention doesn't matter, and the read() or whatever could just return 0 (knowing that the process will never see it), and leave a locked page locked. Things like generic_file_read() could easily use this, and make processes killable even when they are waiting for a hung NFS mount - regardless of any soft mount issues, and without NFS having to have special code.

In the end, I'm too lazy, and I don't care. So I can only tell you how it _should_ be done, and maybe you can tell somebody else until the sucker to actually do it is found.

Linus

深度睡眠不可避免(cont.)

> I'm more in favour of removing TASK_UNINTERRUPTIBLE entirely, or at least
> making people apply for a special licence to be permitted to use it :)

Can't do that.

Easy reason: there are tons of code sequences that _cannot_ take signals. The only way to make a signal go away is to actually deliver it, and there are documented interfaces that are guaranteed to complete without delivering a signal. The trivial case is a disk read: real applications break if you return partial results in order to handle signals in the middle.

In short, this is not something that can be discussed. It's a cold fact, a law of UNIX if you will.

There are enough reasons to discourage people from using uninterruptible sleep ("this f*cking application won't die when the network goes down") that I don't think this is an issue. We need to handle both cases, and while we can expand on the two cases we have now, we can't remove them.

Linus

深度睡眠不可避免(cont.)

>
> Any program setting up signal handlers should expext interrupted i/o,
> otherwise it's buggy.

Roman, THAT IS JUST NOT TRUE!

Go read the standards. Some IO is not interruptible. This is not something
I'm making up, and this is not something that can be discussed about. The
speed of light in vacuum is 'c', regardless of your own relative speed.
And file reads are not interruptible.

Linus

浅度睡眠响应信号

```
static ssize_t globalfifo_read(struct file *filp, char __user *buf,
                               size_t count, loff_t *ppos)
{
        int ret;
        struct globalfifo_dev *dev = container_of(filp->private_data,
                struct globalfifo_dev, miscdev);

        DECLARE_WAITQUEUE(wait, current);

        mutex_lock(&dev->mutex);
        add_wait_queue(&dev->r_wait, &wait);

        while (dev->current_len == 0) {
                if (filp->f_flags & O_NONBLOCK) {
                        ret = -EAGAIN;
                        goto out;
                }
                __set_current_state(TASK_INTERRUPTIBLE);
                mutex_unlock(&dev->mutex);

                schedule();
                if (signal_pending(current)) {
                        ret = -ERESTARTSYS;
                        goto out2;
                }
```

深度睡眠不响应信号

SIGKILL信号也不理会

```
--- a/kernel/drivers/globalfifo/ch12/globalfifo.c
+++ b/kernel/drivers/globalfifo/ch12/globalfifo.c
@@ -114,7 +114,7 @@ static ssize_t globalfifo_read(struct file *filp, char __user *buf,
                        ret = -EAGAIN;
                        goto out;
                }
-               __set_current_state(TASK_INTERRUPTIBLE);
+               __set_current_state(TASK_UNINTERRUPTIBLE);
                mutex_unlock(&dev->mutex);

                schedule();
```

D状态会增加load average

- 使用UNINTERRUPTIBLE
  - ✓ $cat /dev/globalfifo
  - ✓ $cat /dev/globalfifo
- 查看LOAD average：

```
File  Edit  View  Search  Terminal  Help
top - 06:01:47 up  1:06,  5 users,  load average: 2.36, 1.45, 0.64
Tasks: 201 total,   2 running, 199 sleeping,   0 stopped,   0 zombie
%Cpu0  :  8.3 us,  0.3 sy,  0.0 ni, 91.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  6.0 us,  0.7 sy,  0.0 ni, 93.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1023796 total,   935628 used,    88168 free,   150840 buffers
KiB Swap:   522236 total,      240 used,   521996 free.   340336 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2928 baohua    20   0  193992 114484  34256 S  13.6 11.2   2:07.34 compiz
 1825 root      20   0  162644  92084  17104 S   1.3  9.0   0:17.98 Xorg
 2682 root      20   0   27384   2648   2300 S   0.3  0.3   0:00.49 VBoxService
 4209 baohua    20   0  113708  31968  24772 S   0.3  3.1   0:03.91 gnome-term+
 4817 baohua    20   0    6864   3080   2684 S   0.3  0.3   0:00.42 top
```

历史原因

http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html

```
From: Matthias Urlichs <urlichs@smurf.sub.org>
Subject: Load average broken ?
Date: Fri, 29 Oct 1993 11:37:23 +0200


The kernel only counts "runnable" processes when computing the load average.
I don't like that; the problem is that processes which are swapping or
waiting on "fast", i.e. noninterruptible, I/O, also consume resources.

It seems somewhat nonintuitive that the load average goes down when you
replace your fast swap disk with a slow swap disk...

Anyway, the following patch seems to make the load average much more
consistent WRT the subjective speed of the system. And, most important, the
load is still zero when nobody is doing anything. ;-)

--- kernel/sched.c.orig Fri Oct 29 10:31:11 1993
+++ kernel/sched.c  Fri Oct 29 10:32:51 1993
@@ -414,7 +414,9 @@
     unsigned long nr = 0;

     for(p = &LAST_TASK; p > &FIRST_TASK; --p)
-        if (*p && (*p)->state == TASK_RUNNING)
+        if (*p && ((*p)->state == TASK_RUNNING) ||
+                  (*p)->state == TASK_UNINTERRUPTIBLE) ||
+                  (*p)->state == TASK_SWAPPING))
             nr += FIXED_1;
     return nr;
 }
--
Matthias Urlichs        \ XLink-POP N|rnberg   | EMail: urlichs@smurf.sub.org
Schleiermacherstra_e 12  \  Unix+Linux+Mac     | Phone: ...please use email.
90491 N|rnberg (Germany)  \   Consulting+Networking+Programming+etc'ing      42
```

TASK_KILLABLE

- 只响应致命信号

```
@@ -114,11 +114,11 @@ static ssize_t globalfifo_read(struct file *filp, char __user *buf,
                        ret = -EAGAIN;
                        goto out;
                }
-               __set_current_state(TASK_INTERRUPTIBLE);
+               __set_current_state(TASK_KILLABLE);
                mutex_unlock(&dev->mutex);

                schedule();
-               if (signal_pending(current)) {
+               if (fatal_signal_pending(current)) {
                        ret = -ERESTARTSYS;
                        goto out2;
                }
```

其他致命信号可以转化为SIGKILL

致命信号?

```
static inline int __fatal_signal_pending(struct task_struct *p)
{
        return unlikely(sigismember(&p->pending.signal, SIGKILL));
}
```

如果用户态覆盖其他“致命信号”

```c
void sigint(int sig)
{
        printf("got the SIGINT signal...\n");
}

int main(void)
{
        char buf[100];
        int fd=open("/dev/globalfifo",O_RDONLY);

        signal(2, sigint /*SIG_IGN*/);

        if(fd==-1){
                perror("cannot open the FIFO");
                return -11;
        }

        read(fd, buf, 10);

        return 0;
}
```

SIGINT不能再转化为SIGKILL

谢谢!

阅码场出品