Completions

Basic Concurrency Features of the Linux Kernel

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer

Completions

One-way communication between contexts:

- “The requested job is done”
- “An important event has occurred”

```
#include <linux/completion.h>
```

Completing with a semaphore (bad!)

```
foreground_task()
{
  sema_init(&sem);
  start_external_task(&sem);
  down(&sem);
}

interrupt_handler()
{
  up(sem);
  return;
}
```

Completions

Why?

- With semaphores, only one value blocks
- With completions, a block is anticipated

Semaphores are optimized for the opposite use case!

Completions

```
init_completion(struct completion *c)
```

- Initializes a `struct completion`

```
#define DECLARE_COMPLETION(name)
```

- Declares and initializes

Completions

```
void wait_for_completion(struct completion *c)
```

- Uninterruptible wait for completion

```
int wait_for_completion_interruptible(
                    struct completion *c)
```

- Interruptible wait for completion
- Returns nonzero if interrupted

Completions

```
unsigned long
wait_for_completion_interruptible_timeout(
    struct completion *c, unsigned long jiffies)
```

- Interruptible wait for completion, with timeout
- Returns `-ERESTARTSYS` on interruption
- Returns 0 on timeout
- Returns remaining jiffies on completion

Completions

```
void complete(struct completion *c)
```

- Signals completion
- Will wake up only one waiting context

```
void complete_all(struct completion *c)
```

- Signals completion for all waiting contexts

```
#define INIT_COMPLETION(struct completion c)
```

- Reinitializes after `complete_all()`

simple_completion.c

```
1  struct simple_completion_data {
2    struct file_operations fops;
3    struct cdev cdev;
4    struct completion compl;
5  };
6
7  static char stuff[32];
8  static int in, out;
9  #define NSTUFF (sizeof(stuff) / sizeof(*stuff))
```

simple_completion.c

```
1   static ssize_t
2   simple_completion_write (struct file *file, const char __user *buf,
3                            size_t count, loff_t *offp)
4   {
5     int new_in, orig_count = count;
6
7     while(count--) {
8       new_in = in + 1;
9       if (new_in >= NSTUFF) new_in = 0;
10
11      if (new_in != out) {
12          copy_from_user(&stuff[new_in], buf++, 1);
13          in = new_in;
14        }
15      else break;
16    }
17
18    complete(&simple_completion_data.compl);
19    return orig_count;
20  }
```

simple_completion.c

```
1   static ssize_t
2   simple_completion_read (struct file *file, char __user *buf,
3                           size_t count, loff_t *offp)
4   {
5     int bytes = 0;
6
7     if (wait_for_completion_interruptible(&simple_completion_data.compl))
8       return -EINTR;
9
10    while (out != in) {
11      copy_to_user(buf++, &stuff[out], 1);
12      bytes++;
13      if (++out >= NSTUFF) out = 0;
14    }
15
16    return bytes;
17  }
```

`simple_completion_wrong.c`

Question:

- What if we move the `complete()` call?

simple_completion_wrong.c

```
1   static ssize_t
2   simple_completion_write (struct file *file, const char __user *buf,
3                            size_t count, loff_t *offp)
4   {
5     int new_in, orig_count = count;
6
7     while(count--) {
8       new_in = in + 1;
9       if (new_in >= NSTUFF) new_in = 0;
10
11      if (new_in != out) {
12        copy_from_user(&stuff[new_in], buf++, 1);
13        complete(&simple_completion_data.compl);
14        in = new_in;
15      }
16      else break;
17    }
18
19    return orig_count;
20  }
```

kthread-completion.c

```
1   static int kthrd(void *arg)
2   {
3           struct sensor *s = arg;
4           s32 ret;
5
6           while (!kthread_should_stop()) {
7                   init_completion(&s->complete);
8                   wait_for_completion_interruptible(&s->complete);
9
10                  ret = i2c_smbus_read_byte(&s->client);
11                  if (ret >= 0)
12                          pr_err("%s: i2c_smbus_read_byte returned %x\n", __func__, ret);
13          }
14          return 0;
15  }
16
17  static irqreturn_t sensor_interrupt(int irq, void *data)
18  {
19          struct sensor *s = data;
20          complete(&s->complete);
21          return IRQ_HANDLED;
22  }
```

Completions

Basic Concurrency Features of the Linux Kernel

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer