Wait Queues

Conditional Synchronization Between Contexts

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer

Problem Statement

How do I:

- Wake up on a condition, but...
- Sleep otherwise?

Lost Wakeup Problem

```
1   somewhere()
2   {
3           while(1) {
4                   if (!x)
5                           wait_for_completion_interruptible();
6                   /* do something useful... */
7                   x--;
8           }
9   }
10
11  elsewhere()
12  {
13          x++;
14          complete();
15  }
```

Lost Wakeup Problem

The root problem:

- Test-then-act!

Wait Queues

Conditional waits:

- Test condition
- If condition not met, go to sleep
- Repeat

Key difference:

- No preemption during test-then-act

Wait Queues

```
1   do_wait_for_common(struct completion *x, long timeout, int state)
2   {
3           if (!x->done) {
4                   DECLARE_WAITQUEUE(wait, current);
5
6                   __add_wait_queue_tail_exclusive(&x->wait, &wait);
7                   do {
8                           if (signal_pending_state(state, current)) {
9                                   timeout = -ERESTARTSYS;
10                                  break;
11                          }
12                          __set_current_state(state);
13                          spin_unlock_irq(&x->wait.lock);
14                          timeout = schedule_timeout(timeout);
15                          spin_lock_irq(&x->wait.lock);
16                  } while (!x->done && timeout);
17                  __remove_wait_queue(&x->wait, &wait);
18                  if (!x->done)
19                          return timeout;
20          }
21          x->done--;
22          return timeout ?: 1;
23  }
```

Wait Queues

```
1   void complete(struct completion *x)
2   {
3           unsigned long flags;
4
5           spin_lock_irqsave(&x->wait.lock, flags);
6           x->done++;
7           __wake_up_common(&x->wait, TASK_NORMAL, 1, 0, NULL);
8           spin_unlock_irqrestore(&x->wait.lock, flags);
9   }
10  EXPORT_SYMBOL(complete);
```

Wait Queues

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

Wait Queues

`init_waitqueue_head`

- Initializes a `struct wait_queue_head_t`
- Call before any `wake_up()` or wait on the queue

```
void init_waitqueue_head(struct wait_queue_head_t wq);
```

Wait Queues

`DECLARE_WAIT_QUEUE_HEAD`

- Declare-plus-initialize
- Usually appears at file scope

```
#define DECLARE_WAIT_QUEUE_HEAD(wq) ...
```

Wait Queues

```
wait_event(wq, condition)
```

- Waits on `wq` until `condition`

Wait Queues

```
1   #include <linux/wait.h>
2
3   static int x;
4   DECLARE_WAIT_QUEUE_HEAD(wq);
5
6   static void wait_for_x(void)
7   {
8           wait_event(&wq, x != 0);
9   }
10
11  static void trigger_x(void)
12  {
13          x++;
14          wake_up(&wq);
15  }
```

Wait Queues

```
1   static int x;
2   DECLARE_WAIT_QUEUE_HEAD(wq);
3
4   static void wait_for_x1(void)
5   {
6           wait_event(&wq, x == 1);
7   }
8
9   static void wait_for_x2(void)
10  {
11          wait_event(&wq, x == 2);
12  }
13
14  static void trigger_x(void)
15  {
16          x = 1;
17          wake_up(&wq);
18  }
```

Wait Queues

`wake_up(wq)`

- Wakes up all members of `wq` list
- Beware the “thundering herd” problem!

Wait Queues

`wait_event_interruptible(wq, condition)`

- Interruptible wait
- Returns `-ERESTARTSYS` if wait was interrupted

`wait_event_interruptible_timeout(wq, condition, jiffies)`

- Returns `-ERESTARTSYS` if wait was interrupted
- Returns remaining jiffies, or 0 if timeout occurred
- See also `msecs_to_jiffies()`

“Thundering Herd” Problem

What if there are many waiting threads?

- They all wake up on `wake_up()`
- They all check their conditions
- Almost all of them go back to sleep

Can we do better?

“Thundering Herd” Problem

`wait_event_interruptible_exclusive(wq, condition)`

- Waits with `WQ_FLAG_EXCLUSIVE`
- Added to end of wait queue, not beginning
- `wake_up()` stops on first `WQ_FLAG_EXCLUSIVE`

“Thundering Herd” Problem

`wake_up_all(wq)`

- Wakes all wait queue members, regardless of `WQ_FLAG_EXCLUSIVE`

Which One Runs First?

After `wake_up()`:

- All queue members are `TASK_RUNNING`
- Scheduler chooses based on priority
- Use `sched_setscheduler()`, priorities as necessary

Some Clever Code

Look carefully at the following code:

- Why is `xchg()` helpful here?

drivers/rtc/genrtc.c

```
1         if (count != sizeof (unsigned int) && count != sizeof (unsigned long))
2                 return -EINVAL;
3
4         if (file->f_flags & O_NONBLOCK && !gen_rtc_irq_data)
5                 return -EAGAIN;
6
7         retval = wait_event_interruptible(gen_rtc_wait,
8                                           (data = xchg(&gen_rtc_irq_data, 0)));
9         if (retval)
10                goto out;
11
12        /* first test allows optimizer to nuke this case for 32-bit machines */
13        if (sizeof (int) != sizeof (long) && count == sizeof (unsigned int)) {
14                unsigned int uidata = data;
15                retval = put_user(uidata, (unsigned int __user *)buf) ?:
16                        sizeof(unsigned int);
17        }
18        else {
19                retval = put_user(data, (unsigned long __user *)buf) ?:
20                        sizeof(unsigned long);
21        }
```

Wait Queues

Conditional Synchronization Between Contexts

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer