Sleeping in the Kernel

Advanced Embedded Software Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Understand rules for sleeping in the kernel.

Understand kernel sleep/wakeup examples.

Blocking I/O

- How does a driver respond when it can't yet satisfy a request?
  - Block the process, sleeping until the request can proceed
- What does it mean for a process to sleep?
  - Removed from the scheduler run queue
  - Wait for something to change state before running again.

Linux Device Drivers 3rd Edition Chapter 6

Sleeping Rules

- Never sleep when holding a spinlock, seqlock, or RCU lock.
  - Why not?
    - Attempts to obtain spinlocks consume processor resources
      - Other process(es) will consume the processor when attempting to obtain the lock.

Sleeping Rules

- Never sleep if you've disabled interrupts
  - Why not?
    - Interrupt latency would suffer
- Avoid sleep/keep durations short while holding a semaphore/mutex
  - Consider whether you could introduce a deadlock
  - Why is it OK to sleep when holding a semaphore?
    - Other threads waiting for semaphore aren't spinning, they are sleeping

Linux Device Drivers 3rd Edition Chapter 6

Sleeping Rules

- How do you know it's safe to sleep when holding a semaphore/mutex?
  - Ensure any code attempting to obtain the semaphore won't prevent your wake-up condition.
- How do you know this won't happen sometime in the future?
  - You don't, which is why avoiding sleep while holding a semaphore is the best option

Sleeping Rules

- What if I trylock(), see the lock is free, then unlock() sleep and wakeup. Should I assume the lock is still free after wakeup?
  - No - Make no assumptions about the state of system after sleep wakeup
  - Retry locks, if still not available go back to sleep
- Before sleeping, make sure some other process/interrupt/kernel event will wake you up!

Simple Sleeping

```
wait_queue_head_t my_queue;
init_waitqueue_head(&my_queue);
```

```
wait_event(queue, condition)
wait_event_interruptible(queue, condition)
wait_event_timeout(queue, condition, timeout)
wait_event_interruptible_timeout(queue, condition, timeout)
```

- 1st parameter: waitqueue
- 2nd parameter: condition - arbitrary boolean operation
- What do the interruptible versions do?
  - Exit on signals

Simple Sleeping

```
static DECLARE_WAIT_QUEUE_HEAD(wq);
static int flag = 0;

ssize_t sleepy_read (struct file *filp, char __user *buf, size_t count, loff_t *pos)
{
        printk(KERN_DEBUG "process %i (%s) going to sleep\n",
                        current->pid, current->comm);
        wait_event_interruptible(wq, flag != 0);
        flag = 0;
        printk(KERN_DEBUG "awoken %i (%s)\n", current->pid, current->comm);
        return 0; /* EOF */
}
```

- Wouldn't flag != 0 only be evaluated once when calling wait_event_interruptible?
  - Uses a macro to re-evaluate condition parameter

Linux Device Drivers 3rd Edition Chapter 6
https://github.com/cu-ecen-5013/ldd3/blob/latest-lecture-source/misc-modules/sleepy.c#L34

Simple Sleeping

```
wait_event_interruptible(wq, flag != 0);
```

```
/**
 * wait_event_interruptible - sleep until a condition gets true
 * @wq_head: the waitqueue to wait on
 * @condition: a C expression for the event to wait for
 *
 * The process is put to sleep (TASK_INTERRUPTIBLE) until the
 * @condition evaluates to true or a signal is received.
 * The @condition is checked each time the waitqueue @wq_head is woken up.
 *
 * wake_up() has to be called after changing any variable that could
 * change the result of the wait condition.
 *
 * The function will return -ERESTARTSYS if it was interrupted by a
 * signal and 0 if @condition evaluated to true.
 */
#define wait_event_interruptible(wq_head, condition)                    \
({                                                                      \
        int __ret = 0;                                                  \
        might_sleep();                                                  \
        if (!(condition))                                               \
                __ret = __wait_event_interruptible(wq_head, condition); \
        __ret;                                                          \
})
```

```
/*
 * The below macro ___wait_event() has an explicit shadow of the __ret
 * variable when used from the wait_event_*() macros.
 *
 * This is so that both can use the ___wait_cond_timeout() construct
 * to wrap the condition.
 *
 * The type inconsistency of the wait_event_*() __ret variable is also
 * on purpose; we use long where we can return timeout values and int
 * otherwise.
 */

#define ___wait_event(wq_head, condition, state, exclusive, ret, cmd)   \
({                                                                      \
```

```
  init_wait_entry(&__wq_entry, exclusive ? WQ_FLAG_EXCLUSIVE : 0);      \
  for (;;) {                                                            \
          long __int = prepare_to_wait_event(&wq_head, &__wq_entry, state);\
```

```
          if (condition)                                                \
                  break;                                                \
```

```
#define __wait_event_interruptible(wq_head, condition)                  \
        ___wait_event(wq_head, condition, TASK_INTERRUPTIBLE, 0, 0,     \
                      schedule())
```

Linux Device Drivers 3rd Edition Chapter 6
https://elixir.bootlin.com/linux/v5.3.1/source/include/linux/wait.h#L455

Waking Up

```
void wake_up(wait_queue_head_t *queue);
void wake_up_interruptible(wait_queue_head_t *queue);
```

- A different process or interrupt calls wake_up to wake your process up
- First version calls all processes waiting on the specified queue
- Second version calls only interruptible processes on the queue
- Call these when you know your waiters need to wake up

Linux Device Drivers 3rd Edition Chapter 6
https://elixir.bootlin.com/linux/v5.3.1/source/include/linux/wait.h#L208

Waking Up

```
void wake_up(wait_queue_head_t *queue);
void wake_up_interruptible(wait_queue_head_t *queue);
```

- Given a reader and writer queue in a blocking (scull_pipe) design, when would you use wake_up functions?
  - wake_up the reader queue when new data is available due to write
  - wake_up the writer queue when space is available due to reads

Linux Device Drivers 3rd Edition Chapter 6
https://elixir.bootlin.com/linux/v5.3.1/source/include/linux/wait.h#L208