Sleeping and Waking in the Kernel

张雷

zhangl@nju.edu.cn

2018-11-06
runqueue vs wait queues

- The *runqueue* lists group all processes in a TASK_RUNNING state. Processes in a TASK_STOPPED, EXIT_ZOMBIE, or EXIT_DEAD state are not linked in specific lists.

- Processes in a TASK_INTERRUPTIBLE or TASK_UNINTERRUPTIBLE state are subdivided into many classes, each of which corresponds to a specific event. In this case, the process state does not provide enough information to retrieve the process quickly, so it is necessary to introduce additional lists of processes. These are called *wait queues*.
Simple sleeping

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

#define wait_event(wq, condition)
#define wait_event_timeout(wq, condition, timeout)
#define wait_event_interruptible(wq, condition)
#define wait_event_interruptible_timeout(wq, condition, timeout)

#define wake_up(x)
#define wake_up_interruptible(x)
```
wait_event: `<linux/wait.h>`

```c
/**
 * wait_event - sleep until a condition gets true
 * @wq: the waitqueue to wait on
 * @condition: a C expression for the event to wait for
 */
#define wait_event(wq, condition)  
  do { 
  might_sleep(); 
  if (condition) 
    break; 
  __wait_event(wq, condition); 
  } while (0)

#define __wait_event(wq, condition)  
  do { 
  DEFINE_WAIT(__wait); 
  for (;;) { 
    prepare_to_wait(&wq, &__wait, TASK_UNINTERRUPTIBLE); 
    if (condition) 
      break; 
    schedule(); 
  } 
  finish_wait(&wq, &__wait); 
  } while (0)
```
Sleeping via wait queues

A wait queue is a list of processes waiting for an event to occur, wait queues are represented in the kernel by `wake_queue_head_t`.

- created statically via `DECLARE_WAITQUEUE()`
- created dynamically via `init_waitqueue_head()`

Processes put themselves on a wait queue and mark themselves not runnable. When the event associated with the wait queue occurs, the processes on the queue are awakened.

```c
/* 'q' is the wait queue we wish to sleep on */
DEFINE_WAIT(wait);

add_wait_queue(q, &wait);
while (!condition) { /* condition is the event that we are waiting for */
    prepare_to_wait(&q, &wait, TASK_INTERRUPTIBLE);
    if (signal_pending(current))
        /* handle signal */
        schedule(); /*this is where we sleep*/
}
finish_wait(&q, &wait);
```
Sleeping via wait queues: LKD 3ed ch4 p58-61

1. Creates a wait queue entry via the macro DEFINE_WAIT().
2. Adds itself to a wait queue via add_wait_queue(). This wait queue awakens the process when the condition for which it is waiting occurs. There needs to be code elsewhere that calls wake_up() on the queue when the event actually does occur.
3. Calls prepare_to_wait() to change the process state to either TASK_INTERRUPTIBLE or TASK_UNINTERRUPTIBLE. This function also adds the task back to the wait queue if necessary, which is needed on subsequent iterations of the loop.
4. If the state is set to TASK_INTERRUPTIBLE, a signal wakes the process up. This is called a spurious wake up (a wake-up not caused by the event occurrence). So check and handle signals.
5. When the task awakens, it again checks whether the condition is true. If it is, it exits the loop. Otherwise, it again calls schedule() and repeats.
6. After the condition is true, the task sets itself to TASK_RUNNING and removes itself from the wait queue via finish_wait().