Introduction to Work Queues

Deferred Kernel Work

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer

Overview

Roadmap:

- What is a “workqueue”?
- `struct work_struct`
- `queue_work()`
- `schedule_work()`
- Examples

Overview

Roadmap:

- Creating your own workqueue
- `create_workqueue()`
- `create_singlethread_workqueue()`
- Controlling queued work

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

Defining a “Workqueue”

Workqueue:

- Linked list of work queue entries
- Each entry is a “work function”

Like tasklets, but:

- Work functions run in a process context
- Work functions can sleep, etc.

Defining a “Workqueue”

Work queue entries:

- Cannot be scheduled more than once (see also tasklets)
- Can be assigned to a default queue, or one of your own

Uses:

- Long-running, kernel-context things
- Fire-and-forget
- Alternative (maybe) to kernel threads

Defining a “Workqueue”

Alternative to kernel threads:

- Fire-and-forget
- Queuing logic might eliminate loops
- No need for `kthread_should_stop()`

But:

- Slightly more overhead to initiate
- Won’t accumulate run requests (see also completions)

Creating Work Queue Entries

Declare-then-initialize:

- Two-step process

```
struct work_struct work;
void work_function(struct work_struct *work);

INIT_WORK(&work, work_function);
```

Creating Work Queue Entries

Declare-and-initialize:

- A single step, but less clear (to me, anyway)

```
void work_function(struct work_struct *work);

/* struct work_struct work; */
static DECLARE_WORK(work, work_function);
```

Creating Work Queue Entries

Changing the work function:

- Work queue item must already be properly initialized!
- Safe to call on already-scheduled work (!)

```
void new_work_function(struct work_struct *work);

PREPARE_WORK(&work, new_work_function);
```

Scheduling Work

There exists a “default” work queue:

- Shared resource
- Ideal for non-time-critical work

```
struct work_struct work;
...

schedule_work(&work);
```

Scheduling Work

Scheduling delayed work:

- Work begins no earlier than the specified time
- Ideal for holdoff timers
- Work can be canceled before it begins

```
#define DELAY (100 * HZ/1000) /* 100 msecs */
struct work_struct work;
...

schedule_delayed_work(&work, DELAY);
```

Canceling Work

Why?

- Evasive action
- Kernel module unlinking

Canceling Work

Canceling queued work:

- Returns 1 if the work wasn't already running
- Otherwise, blocks until work is complete and returns 0
- Will prevent work function from re-scheduling itself

```
cancel_work_sync(&work);
```

Canceling Work

Canceling delayed queued work:

- Returns 1 if the work wasn't already running
- Returns 0 otherwise, prevents work function from re-scheduling
- May return before work function finishes

```
cancel_delayed_work(&work);
```

Canceling Work

Canceling delayed work, with sync:

- Returns 1 if the work wasn't already running
- Returns 0 otherwise, prevents work function from re-scheduling
- Blocks until work function finishes

```
cancel_delayed_work_sync(&work);
```

Waiting for Work to Finish

"Flushing" a work queue:

- Blocks until all work functions have finished

```
flush_scheduled_work(void);
```

Waiting for Work to Finish

Waiting for a specific work item:

- Blocks, returns 1 when the work function exits
- Returns 0 if work item wasn't queued

```
flush_work(&work);
```

Custom Work Queues

Useful for:

- Specialized prioritization needs
- Don’t want to pollute generic work queue

Variations:

- Non-reentrant
- Freezeable
- High-priority
- Generic

Custom Work Queues

Generic work queue:

- Works queue entries in sequential order
- Equivalent to the global work queue

```
/* struct workqueue_struct generic_workqueue; */
create_workqueue(generic_workqueue);
```

Custom Work Queues

"Freezeable" work queue:

- Work functions are paused during suspends

```
/* struct workqueue_struct freezable_workqueue; */
create_freezeable_workqueue(freezeable_workqueue);
```

Custom Work Queues

Non-reentrant work queue:

- Guarantees that the work function isn't reentered

```
/* struct workqueue_struct nonreentrant_workqueue; */
alloc_workqueue(nonreentrant_workqueue,
                WQ_NON_REENTRANT, 1);
```

Custom Work Queues

High-priority work queue:

- Workers are added at the beginning of the list, rather than end

```
/* struct workqueue_struct highprio_workqueue; */
alloc_workqueue(highprio_workqueue,
                WQ_HIGHPRI, 1);
```

Custom Work Queues

Queueing work to private queues:

- Delayed and immediate versions supported
- Both return 0 if the work item was already queued

```
int queue_work(struct workqueue_struct *wq,
    struct work_struct *work);
int queue_delayed_work(struct workqueue_struct *wq,
    struct work_struct *work, unsigned long jiffies);
```

Custom Work Queues

Queueing work to private queues:

- Delayed and immediate versions supported
- Both return 0 if the work item was already queued

```
int queue_work(struct workqueue_struct *wq,
    struct work_struct *work);
int queue_delayed_work(struct workqueue_struct *wq,
    struct work_struct *work, unsigned long jiffies);
```

One Last Word...

Note:

- Work queue implementation completely overhauled in 2.6.36!

Introduction to Work Queues

Deferred Kernel Work

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer