* Kernel stack for each process
  \[ \rightarrow 4k \text{ or } 8k \]
  * Linked list of task-struct
    \[ \rightarrow \text{"tasks"} \]
  * At the bottom of each process's kernel stack, you have "thread-info" struct.

```
movl $-8192, %eax
andl %esp, %eax
```
The proc is in RQ, but not running on CPU.

wake_up()
→ try_wake_up()
→ sets T-R
put the proc back into RQ

preemption
→ by the timer interrupt, for example.

prepare_to_wait()
→ set T-I or T-U

schedule()
→ context_switch()

The proc is running on CPU.

Signal received
→ Proc is put back to RQ; and when it runs, it will run sig handler.
Wait queues (pseudo code)

```c
struct wg_head {
    spin_lock_t lock;
    list_head task_list;
};

struct wg {
    task_struct *task;
    wg_func func; // call back function
    list_head task_list;
};
```

```c
void wait_event(wg, cond) {
    DEF_WAIT(wait);
    for (;;) {
        prepare_to_wait(
            &wg,
            &wait,
            T_I
        );
        if (cond) break;
        schedule();
    }
}
```

See LKD p59 for explanation

"wait-event" is a real function in Linux kernel. Skeleton pseudo code is shown here.
Presumption

* Read LKD 62 - 64 (before Real-Time Sched Policy)
  → See the cases when "need_resched" flag is checked.

Kernel Spin-lock

* LKD chapter 10: page 183 - 202
  → We didn't get to cover this in the lectures, so this is optional.

Interrupts

* LKD chapter 7:
  → We covered it at a high level, but omitted details found on chapter 7. So it's optional.