* kernel stack for each process
  
  - 4k or 8k
  
  * linked list at task_struct
  
  - "tasks"

  * At the bottom of each process's kernel stack, you have "thread_info" struct.

```
movl $-8192,%eax
andl %esp,%eax
```}

- Enables quick to your own task_struct.

- Physical RAM

- Program break

- Unmapped region

- proc 1

- proc 2

- BRK

- heap

- static

- text

- Stack

- BRK

- text

- static

- heap

- Stack

- kernel code & data

- kernel code & data

- 4G

- 4G

- 3G

- 3G

- Brk

- physical RAM
The proc is in RQ, but not running on CPU.

- \texttt{wake\_up()}, \texttt{try\_to\_wake\_up()}
- \texttt{set T-R}
  - put the proc back into RQ

\texttt{preemption} by the timer interrupt, for example.

\texttt{prepare\_to\_wait()}
- \texttt{set T-I or T-U}
- \texttt{schedule()}
- \texttt{take the proc out of the RQ.}

\texttt{signal received}
- Proc is put back to RQ; and when it runs, it will run sig handler.

\textbf{Fig. 3.3}
\textbf{P.28}
\textbf{LkD}

\textbf{States}
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();
    }
    finish_wait(&wg, &wait);
}
```

"wait_event" is a real function in Linux kernel.
Skeleton Pseudo code is shown here.