以前pid是一个global的整数，而现在pid变成了一个structure (???)

in include/linux/pid.h

```
/*
 * What is struct pid?
 *
 * A struct pid is the kernel's internal notion of a process identifier.
 * It refers to individual tasks, process groups, and sessions.  While
 * there are processes attached to it the struct pid lives in a hash
 * table, so it and then the processes that it refers to can be found
 * quickly from the numeric pid value.  The attached processes may be
 * quickly accessed by following pointers from struct pid.
 *
 * Storing pid_t values in the kernel and referring to them later has a
 * problem.  The process originally with that pid may have exited and the
 * pid allocator wrapped, and another process could have come along
 * and been assigned that pid.
 *
 * Referring to user space processes by holding a reference to struct
 * task_struct has a problem.  When the user space process exits
 * the now useless task_struct is still kept.  A task_struct plus a
 * stack consumes around 10K of low kernel memory.  More precisely
 * this is THREAD_SIZE + sizeof(struct task_struct).  By comparison
```

```
 * a struct pid is about 64 bytes.
 *
 * Holding a reference to struct pid solves both of these problems.
 * It is small so holding a reference does not consume a lot of
 * resources, and since a new struct pid is allocated when the numeric pid
 * value is reused (when pids wrap around) we don't mistakenly refer to new
 * processes.
 */



/*
 * struct upid is used to get the id of the struct pid, as it is
 * seen in particular namespace. Later the struct pid is found with
 * find_pid_ns() using the int nr and struct pid_namespace *ns.
 */

struct upid {
        /* Try to keep pid_chain in the same cacheline as nr for find_vpid */
        int nr;
        struct pid_namespace *ns;
        struct hlist_node pid_chain;
};

struct pid
{
```

```
        atomic_t count;
        unsigned int level;
        /* lists of tasks that use this pid */
        struct hlist_head tasks[PIDTYPE_MAX];
        struct rcu_head rcu;
        struct upid numbers[1];
};
```

怎样获得当前process的pid (struct) ?

参考cad_pid

in kernel/reboot.c

cad means ctrl-alt-del

in init/main.c

kernel_init_freeable()

```
cad_pid = task_pid(current);
==>
#define current get_current()
==>
```

#define get_current() (current_thread_info()->task)

==>

static inline struct thread_info *current_thread_info(void)

{

register unsigned long sp asm ("sp");

return (struct thread_info *)(sp & ~(THREAD_SIZE - 1));

}

1. 取得当前kernel stack的sp

2. 目前ARM Linux kernel platform，kernel stack size = 8K (2 pages)

8K kernel stack的bottom是struct thread_info.

in arch/arm/include/asm/thread_info.h

```
/*
 * low level task data that entry.S needs immediate access to.
 * __switch_to() assumes cpu_context follows immediately after cpu_domain.
 */
struct thread_info {
        unsigned long           flags;          /* low level flags */
        int                     preempt_count;  /* 0 => preemptable, <0 => bug */
        mm_segment_t            addr_limit;     /* address limit */
        struct task_struct      *task;          /* main task structure */
        struct exec_domain      *exec_domain;   /* execution domain */
        __u32                   cpu;            /* cpu */
        __u32                   cpu_domain;     /* cpu domain */
        struct cpu_context_save cpu_context;    /* cpu context */
        __u32                   syscall;        /* syscall number */
        __u8                    used_cp[16];    /* thread used copro */
        unsigned long           tp_value[2];    /* TLS registers */
#ifdef CONFIG_CRUNCH
        struct crunch_state     crunchstate;
#endif
        union fp_state          fpstate __attribute__((aligned(8)));
        union vfp_state         vfpstate;
#ifdef CONFIG_ARM_THUMBEE
        unsigned long           thumbee_state;  /* ThumbEE Handler Base register
*/
#endif
        struct restart_block    restart_block;
};
```

3. 由thread_info->task找到代表process的struct task_struct。

4. 由struct task_struct而得到pid

```
static inline struct pid *task_pid(struct task_struct *task)
{
    return task->pids[PIDTYPE_PID].pid;
}
```