进程是什么

进程是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位。

- 它有自己的目标，或者说功能，同时又能受控于进程调度模块；
- 它可以利用系统的资源，有自己的代码和数据，同时拥有自己的堆栈；
- 进程需要被调度。

进程表是什么

进程表是存储进程状态信息的数据结构`PUBLIC PROCESS proc_table[NR_TASKS];`

进程表是进程存在的唯一标识，是操作系统用来记录和刻画进程状态及环境信息的数据结构，是进程动态特征的汇集，也是操作系统掌握进程的唯一资料结 构和管理进程的主要依据。

进程栈是什么

进程运行时自身的堆栈，包括用户栈和核心栈

当寄存器的值已经被保存到进程表内，esp应指向何处来避免破坏进程表的值

esp 指向专门的内核栈区域

```
save:
        pushad          ; `.
        push    ds      ;  |
        push    es      ;  | 保存原寄存器值
        push    fs      ;  |
        push    gs      ; /
        mov     dx, ss
        mov     ds, dx
        mov     es, dx

        mov     esi, esp                    ;esi = 进程表起始地址

        inc     dword [k_reenter]           ;k_reenter++;
        cmp     dword [k_reenter], 0        ;if(k_reenter ==0)
        jne     .1                          ;{
        mov     esp, StackTop               ;  mov esp, StackTop <--切换到内核栈
        push    restart                     ;  push restart
        jmp     [esi + RETADR - P_STACKBASE];  return;
.1:                                         ;} else { 已经在内核栈，不需要再切换
        push    restart_reenter             ;  push restart_reenter
        jmp     [esi + RETADR - P_STACKBASE];  return;
                                            ;}
```

tty是什么

原来是指电传打字机，也**泛指计算机的终端(terminal)设备**

早期的终端(terminal) 是一台独立于计算机的机器(teletype 即, TTY)

终端

不同的tty为什么输出不同的画面在同一个显示器上

不同 TTY 各有一个 CONSOLE，各个 CONSOLE 公用同一块显存。

虽然不同的 TTY 对应的输入设备是同一个键盘，但输出却好比是在不同的显示器上，因为不同的 TTY 对应的屏幕画面可能是迥然不同的。实际上，我们当然是在使用同一个显示器，画面的不同只不过是因为显示了显存的不同位置罢了

显示了显存的不同位置

解释tty任务执行的过程

在 TTY 任务中执行一个循环，这个循环将轮询每一个 TTY，处理它的事件， 包括从键盘缓冲区读取数据、显示字符等内容。

```
for (p_tty = TTY_FIRST; p_tty < TTY_END; p_tty++)
{
    tty_do_read(p_tty);
    tty_do_write(p_tty);
}
```

轮询到每一个 TTY 时：

- 处理输入：查看其是否为当前 TTY。只有当某个 TTY 对应的控制台是当前控制台时，它才可以读取键盘缓冲区。

```
PRIVATE void tty_do_read(TTY *p_tty)
{
    if (is_current_console(p_tty->p_console))
    {
        keyboard_read(p_tty);
    }
}
```

`keyborad_read` → `in_process` → `put_key`

- 处理输出：如果有要显示的内容则显示它

```
PRIVATE void tty_do_write(TTY *p_tty)
{
    if (p_tty->inbuf_count)
    {
        char ch = *(p_tty->p_inbuf_tail);
        p_tty->p_inbuf_tail++;
        if (p_tty->p_inbuf_tail == p_tty->in_buf + TTY_IN_BYTES)
        {
            p_tty->p_inbuf_tail = p_tty->in_buf;
        }
        p_tty->inbuf_count--;
        push_out_char(p_tty->p_console,ch); // 新增，压入操作记录栈
        out_char(p_tty->p_console, ch);
    }
}
```

`out_char(CONSOLE* p_con, char ch)`

tty结构体中大概包括哪些内容

```
/* TTY */
typedef struct s_tty
{
    u32 in_buf[TTY_IN_BYTES];   /* TTY 输入缓冲区 */
    u32*    p_inbuf_head;       /* 指向缓冲区中下一个空闲位置 */
    u32*    p_inbuf_tail;       /* 指向键盘任务应处理的键值 */
    int inbuf_count;        /* 缓冲区中已经填充了多少 */

    struct s_console *  p_console;
}TTY;
```

输入缓存区及其头尾指针和填充的大小

指向console的指针

console结构体中大概包括哪些内容

```
/* CONSOLE */
typedef struct s_console
{
    unsigned int    current_start_addr; /* 当前显示到了什么位置      */
    unsigned int    original_addr;      /* 当前控制台对应显存位置 */
    unsigned int    v_mem_limit;        /* 当前控制台占的显存大小 */
    unsigned int    cursor;             /* 当前光标位置 */
    unsigned int    search_start_pos;   /*新增：ESC模式开始时候的cursor位置*/
    POSSTACK pos_stack;                 /*新增*/
    OUTCHARSTACK out_char_stack;        /*新增*/
}CONSOLE;
```

当前显示到的位置、光标位置

当前控制台对应的显存位置和占用的大小

什么是时间片

时间片即CPU分配给各个程序的时间，每个线程被分配一个时间段，称作它的时间片，即该进程允许运行的时间，使各个程序从表面上看是同时进行的。