Day 17

如果所有任务都去休眠了怎么办？系统无事可做，我们只好hlt。为了降低编码复杂度，我们可以设置一个idle process，他的功能就是循环hlt

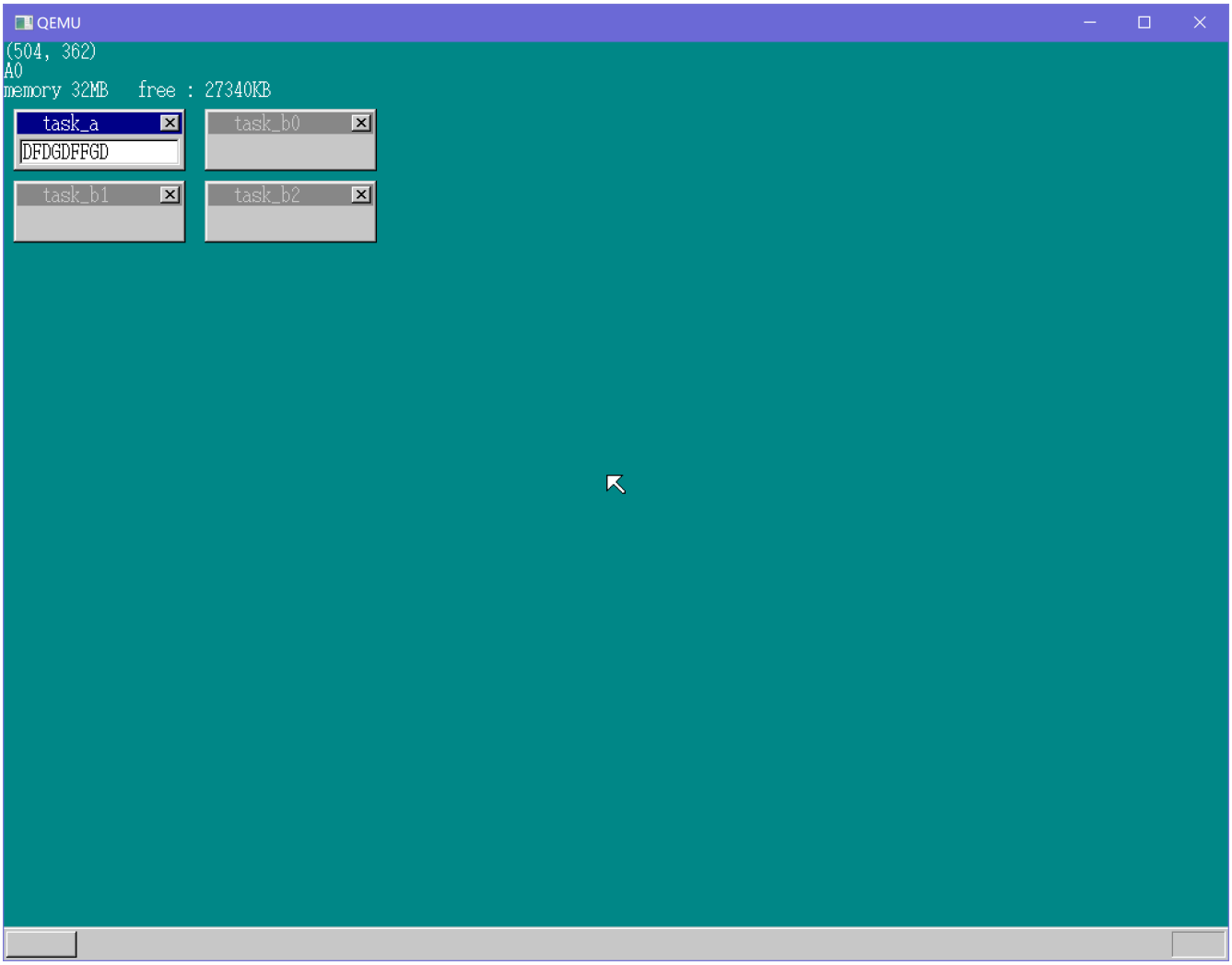
```
for (;;) io_hlt();
```

这个idle process具有最低的优先级，他就像链表中的哨兵一样，是为了降低我们编码复杂度而存在。

我们所要接着做的，就是再task_init的时候，把这个task放入最下层的taskctl中。

```
struct TASK *task_init(struct MEMMAN *memman)
{
    struct TASK *task, *idle;
    idle = task_alloc();
    idle->tss.esp = memman_alloc_4k(memman, 64 * 1024) + 64 * 1024;
    idle->tss.eip = (int) &task_idle;
    idle->tss.es = 1 * 8;
    idle->tss.cs = 2 * 8;
    idle->tss.ss = 1 * 8;
    idle->tss.ds = 1 * 8;
    idle->tss.fs = 1 * 8;
    idle->tss.gs = 1 * 8;
    task_run(idle, MAX_TASKLEVELS - 1, 1);
    return task;
}
```

我们修改一下我们的程序，不启动taskb，只保留输入框的taska，然后我们不给外部输入。



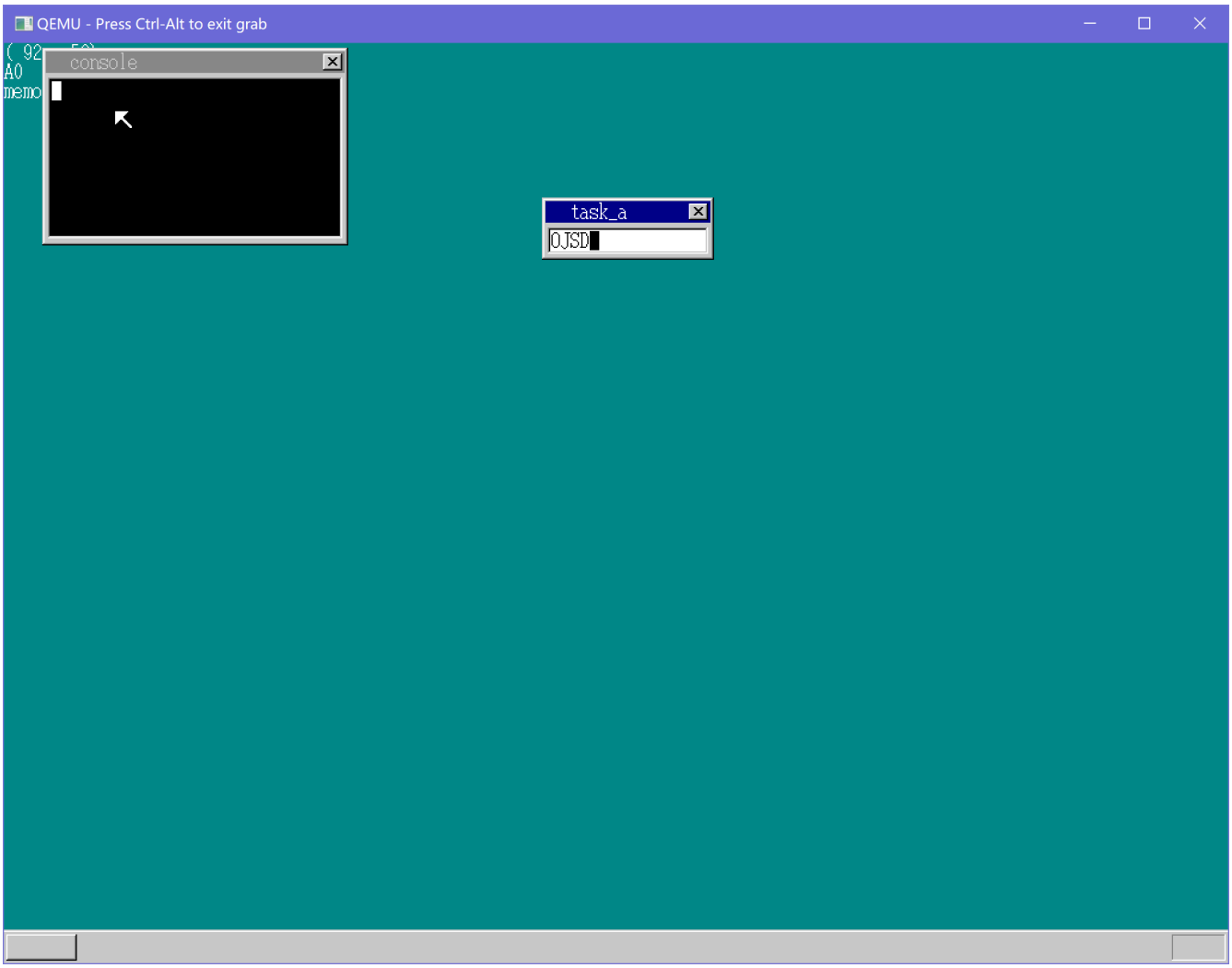
工作没有出现异常

然后我们来创建命令行窗口，方法是和创建输入框大同小异的，不同的是我们再让自己休眠的时候可以用task_now()函数来获取自己task的地址

```
void console_task(struct SHEET *sheet)
{
    struct FIFO32 fifo;
    struct TIMER *timer;
    struct TASK *task = task_now();

    int i, fifobuf[128], cursor_x = 8, cursor_c = COL8_000000;
    fifo32_init(&fifo, 128, fifobuf, task);

    timer = timer_alloc();
    timer_init(timer, &fifo, 1);
    timer_settime(timer, 50);

    for (;;) {
        io_cli();
```

```
        if (fifo32_status(&fifo) == 0) {
            task_sleep(task);
            io_sti();
        } else {
            i = fifo32_get(&fifo);
            io_sti();
            if (i <= 1) {
                if (i != 0) {
                    timer_init(timer, &fifo, 0);
                    cursor_c = COL8_FFFFFF;
                } else {
                    timer_init(timer, &fifo, 1);
                    cursor_c = COL8_000000;
                }
                timer_settime(timer, 50);
                boxfill8(sheet->buf, sheet->bxsize, cursor_c, cursor_x, 28, cursor_x + 7,
43);
                sheet_refresh(sheet, cursor_x, 28, cursor_x + 8, 44);
            }
        }
    }
}
```

make run嘻嘻

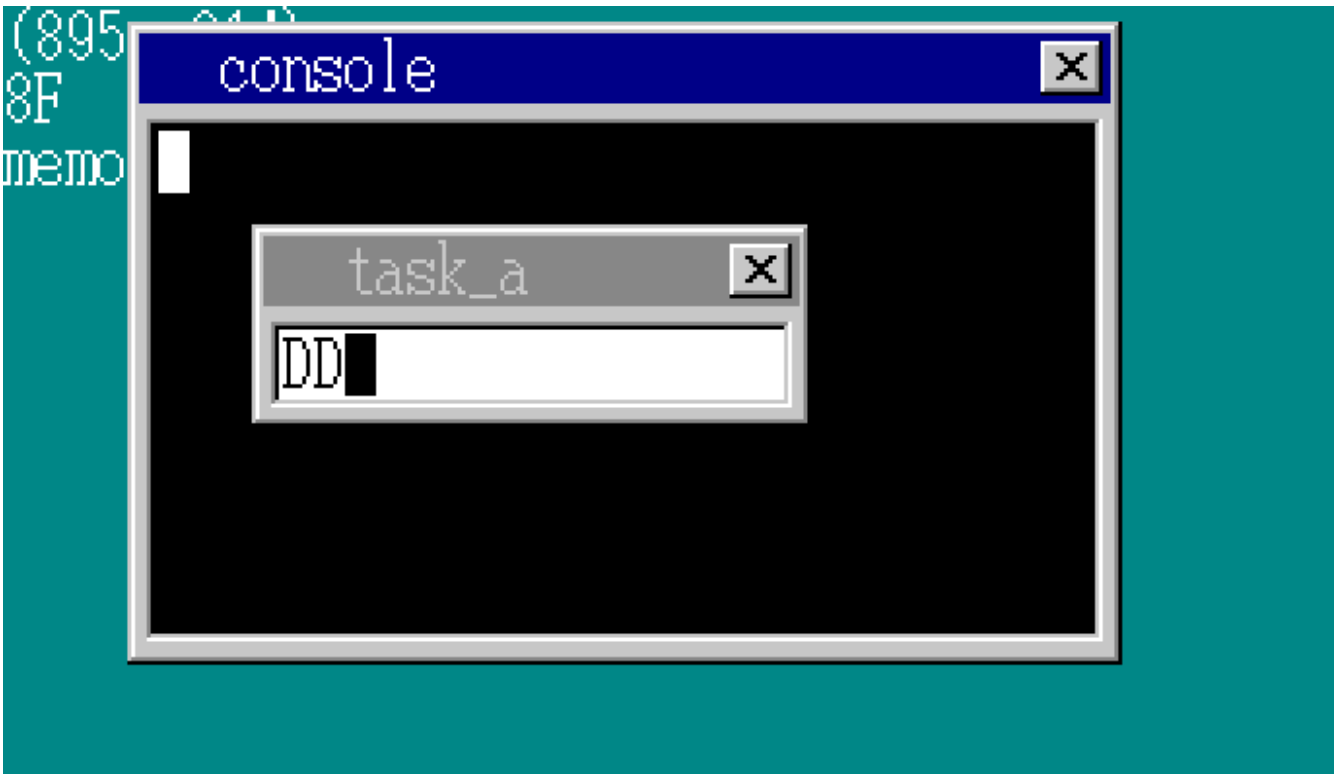


两个光标都在闪烁，但是相位稍有差别，看来是启动时间不一致所导致的。

之前我们只能再taska的输入框中进行输入，我们要想再console中输入，我们必须要实现切换窗口焦点的功能，我们决定使用tab键作为切换窗口焦点的快捷键。窗口获得焦点之后，窗口标题栏应当变为蓝色，而非焦点窗口标题栏应当变为灰色。

我们再设置一个keyto变量，用于指定焦点在哪一个窗体，之后我们处理按键事件的时候就可以根据keyto变量做出相应的响应

```
if (i == 256 + 0x0f) { /* 按键事件处理中对tab的处理 */
    if (key_to == 0) {
        key_to = 1;
        make_wtitle8(buf_win,  sht_win->bxsize,  "task_a",  0);
        make_wtitle8(buf_cons, sht_cons->bxsize, "console", 1);
    } else {
        key_to = 0;
        make_wtitle8(buf_win,  sht_win->bxsize,  "task_a",  1);
        make_wtitle8(buf_cons, sht_cons->bxsize, "console", 0);
    }
    sheet_refresh(sht_win,  0, 0, sht_win->bxsize,  21);
    sheet_refresh(sht_cons, 0, 0, sht_cons->bxsize, 21);
}
```

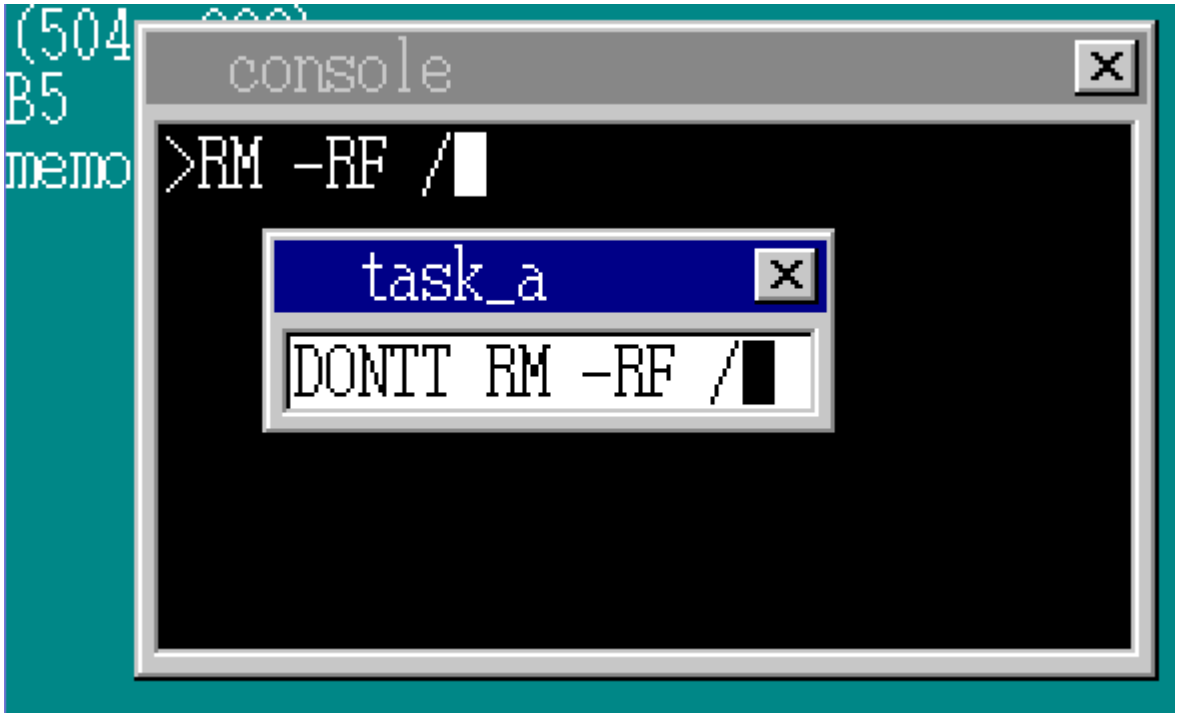


为了让输入也能够在窗体之间切换，我们还需要做些其他的事情

为了在console中输入，我们要知道console的fifo，然后只要把数据送到哪个fifo里就可以了。在task中添加一个fifo，让他作为按键事件的fifo。注意我们送进去ascii而不是原始十六进制数，这样可以不用单独的在consoletask中写转换逻辑了。

然后我们在console task中写上相应的按键事件处理逻辑。

```
void console_task(struct SHEET *sheet)
{
    struct TIMER *timer;
    struct TASK *task = task_now();
    int i, fifobuf[128], cursor_x = 16, cursor_c = COL8_000000;
    char s[2];

    fifo32_init(&task->fifo, 128, fifobuf, task);
    timer = timer_alloc();
    timer_init(timer, &task->fifo, 1);
    timer_settime(timer, 50);
    putfonts8_asc_sht(sheet, 8, 28, COL8_FFFFFF, COL8_000000, ">", 1); // 提示符
    for (;;) {
        io_cli();
        if (fifo32_status(&task->fifo) == 0) {
            task_sleep(task);
            io_sti();
        } else {
            i = fifo32_get(&task->fifo);
            io_sti();
            if (i <= 1) { // 光标处理
                if (i != 0) {
                    timer_init(timer, &task->fifo, 0);
                    cursor_c = COL8_FFFFFF;
                } else {
                    timer_init(timer, &task->fifo, 1);
                    cursor_c = COL8_000000;
                }
                timer_settime(timer, 50);
            }
            if (256 <= i && i <= 511) { // 来自task a的键盘数据
                if (i == 8 + 256) { // 退格键
                    if (cursor_x > 16) {
                        putfonts8_asc_sht(sheet, cursor_x, 28, COL8_FFFFFF, COL8_000000, "
", 1);
                        cursor_x -= 8; // 擦除前移
                    }
                } else { // 一般字符
                    if (cursor_x < 240) {
                        s[0] = i - 256; // 打印字符并后移
                        s[1] = 0;
                        putfonts8_asc_sht(sheet, cursor_x, 28, COL8_FFFFFF, COL8_000000, s,
1);
                        cursor_x += 8;
                    }
                }
            }
            boxfill8(sheet->buf, sheet->bxsize, cursor_c, cursor_x, 28, cursor_x + 7, 43);
            sheet_refresh(sheet, cursor_x, 28, cursor_x + 8, 44);
        }
    }
}
```



工作正常

接下来我们处理shift键，当shift键按下的时候，应当改变默认的大小写输入状态，若是符号或者数字键，则应当输入上方的符号。

我们准备一个key_shift变量，当左Shift按下时置为1，右Shift按下时置为2，两个都不 按时置为0，两个都按下的时候就置为3。

```
if (i == 256 + 0x2a) {  // lshift on
    key_shift |= 1;
}
if (i == 256 + 0x36) {  // rshift on
    key_shift |= 2;
}
if (i == 256 + 0xaa) {  // lshift off
    key_shift &= ~1;
}
if (i == 256 + 0xb6) {  // rshift off
    key_shift &= ~2;
}
```

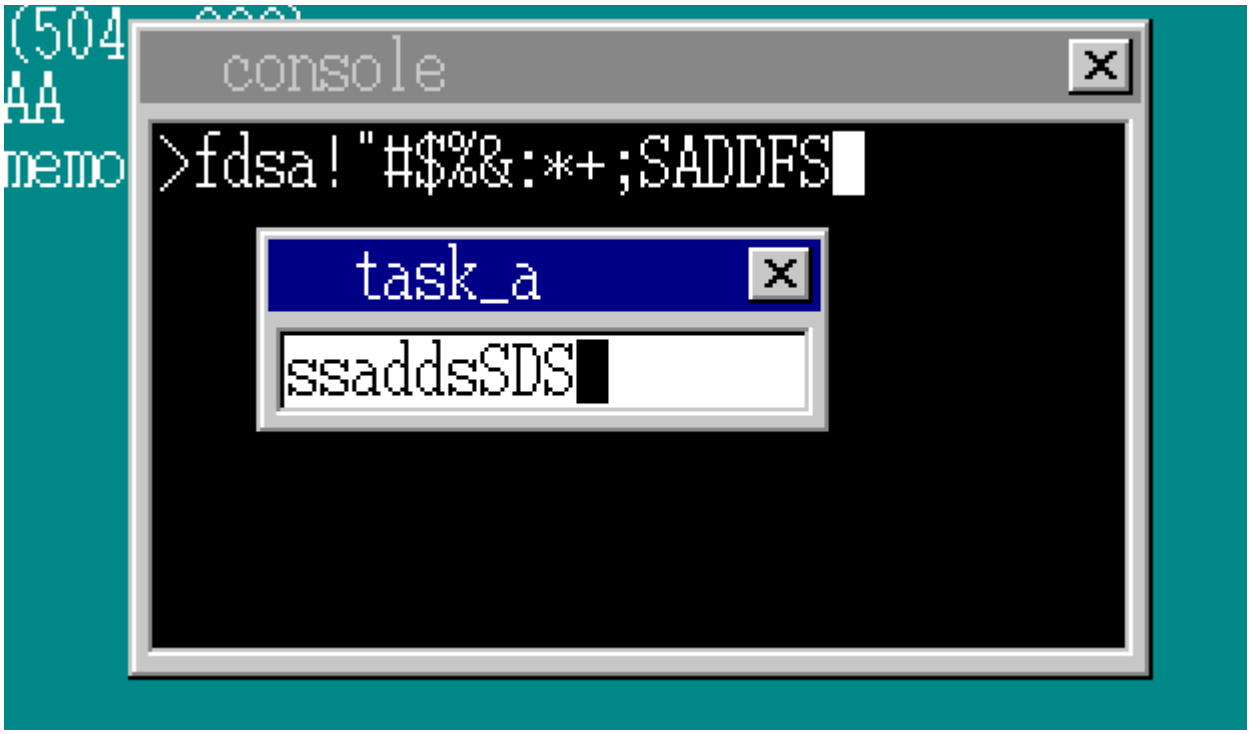
除此之外我们还需要处理一下caps lock，从key_leds中取出指定的bit就可以了。

binfo->leds 的第 4 位 → ScrollLock 状态 binfo->leds 的第 5 位 → NumLock 状态 binfo->leds 的第 6 位 → CapsLock 状态

然后输入大写的条件是key_shift^caps_lock

```
if ('A' <= s[0] && s[0] <= 'Z') {
    if (((key_leds & 4) == 0 && key_shift == 0) ||
    ((key_leds & 4) != 0 && key_shift != 0)) {
        s[0] += 0x20;
    }
}
```

make run



咦？模拟器中caps lock怎么不好用呢？

原来亮灯灭灯逻辑需要我们自己来写

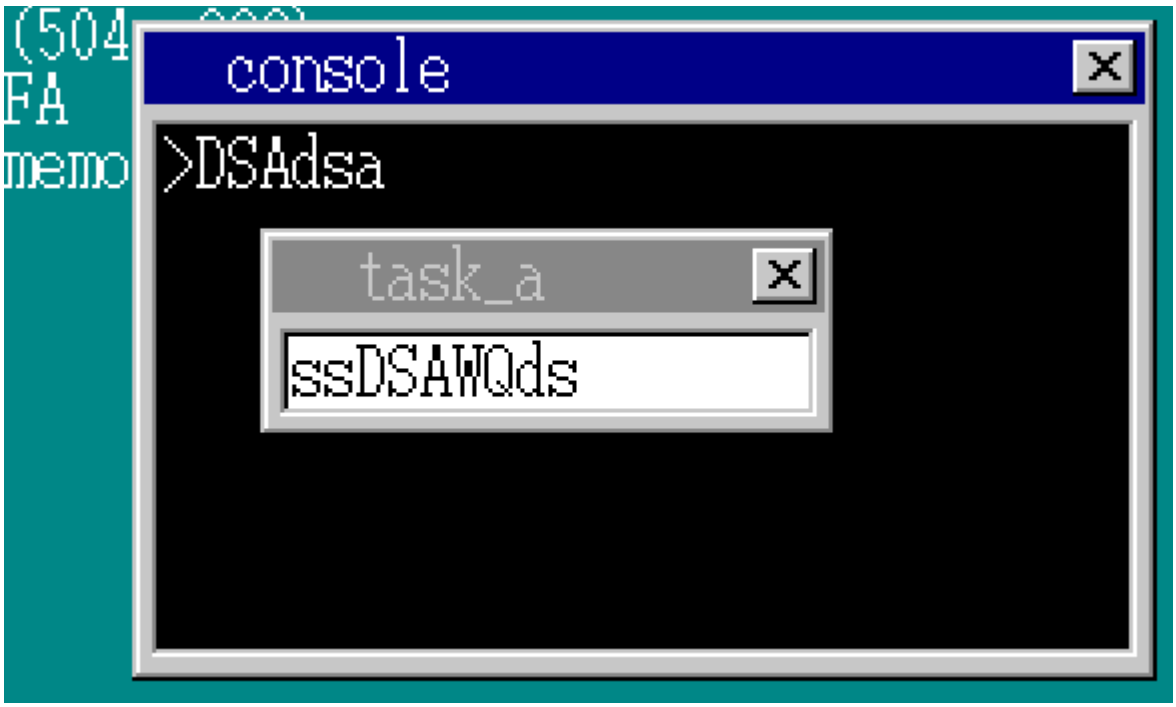
- 读取状态寄存器，等待 bit 1的值变为 0。
- 向数据输出（0060）写入要发送的 1个字节数据。
- 等待键盘返回 1个字节的信息，这和等待键盘输入所采用的方法相同（用 IRQ等待 或者用轮询状态寄存器 bit 1 的值直到其变为 0都可以）。
- 返回的信息如果为 0xfa，表明 1个字节的数据已成功发送给键盘。如为 0xfe则表明 发送失败，需要返回第 1步重新发送。

而要控制LED的状态，需要按上述方法执行两次，向键盘发送EDxx数据。其中，xx的bit0代表ScrollLock，bit 1代表NumLock，bit 2代表CapsLock（0表示熄灭，1表示点亮）。bit 3～7为保留位，置0即可

```
for (;;) {
    if (fifo32_status(&keycmd) > 0 && keycmd_wait < 0) { /*从此开始*/
        /*如果存在向键盘控制器发送的数据，则发送它 */
        keycmd_wait = fifo32_get(&keycmd);
        wait_KBC_sendready();
        io_out8(PORT_KEYDAT, keycmd_wait);
    } /*到此结束*/
```

```
    io_cli();
    if (fifo32_status(&fifo) == 0) {
        task_sleep(task_a);
        io_sti();
    } else {
        i = fifo32_get(&fifo);
        io_sti();
        if (256 <= i && i <= 511) { /* 键盘数据 */
            /*从此开始*/ if (i == 256 + 0x3a) { /* CapsLock */
                key_leds ^= 4;
                fifo32_put(&keycmd, KEYCMD_LED);
                fifo32_put(&keycmd, key_leds);
            }
            if (i == 256 + 0x45) { /* NumLock */
                key_leds ^= 2;
                fifo32_put(&keycmd, KEYCMD_LED);
                fifo32_put(&keycmd, key_leds);
            }
            if (i == 256 + 0x46) { /* ScrollLock */
                key_leds ^= 1;
                fifo32_put(&keycmd, KEYCMD_LED);
                fifo32_put(&keycmd, key_leds);
            }
            if (i == 256 + 0xfa) { /*键盘成功接收到数据*/
                keycmd_wait = -1;
            }
            if (i == 256 + 0xfe) { /*键盘没有成功接收到数据*/
                wait_KBC_sendready();
                io_out8(PORT_KEYDAT, keycmd_wait);
                /*到此结束*/ }
        } else if (512 <= i && i <= 767) { /*鼠标数据*/

        } else if (i <= 1) { /*光标用定时器*/

        }
    }
}
```

make run一下



完美