NNOS API文档（第〇版，nnos_34d）

> 更新日期：2020-06-30
>
> NNOS项目：github地址
>
> NNOS开发者交流群： 

一、简介

1.1 NNOS简介

- NNOS（下称**本系统**）是一款基于Intel 80x86架构的操作系统；
- NNOS，即“年年操作系统”，为临时名称，尚未正式命名；
- 本系统共五层（引导层、基础层、核心层、扩展层、应用层）；
- 本系统包含四大功能：内存管理、进程管理、文件管理、I/O管理；
- 本系统已通过GPL协议开源；
- 详情请参照NNOS参考手册（单击下载）。

1.2 关于本文档

- 本文档提供给NNOS系统开发人员使用；
- 本文档由全体开发人员维护；
- 本文档为NNOS所有函数接口之功能、用法的详细说明；
- 本文档仅适用于标称版本的系统开发，随系统更新而同步更新；
- 如有任何疑问，欢迎在[NNOS开发者交流群]交流。

1.3 凡例

- 出于习惯，本文档通常直接称API为函数，只有在需要区分API与普通函数的语境下，才使用`API`一词；
- 斜体*API*为未实现或本系统未使用的函数,如：*io_in16()*；
- 带删除线的~~API~~为已废弃或不建议继续使用的函数，如：~~io_hlt()~~；
- 本文档所有涉及度量(如长度、宽度、高度)处，除特别标明外，皆以像素为单位。

二、系统地图

> 本章为系统层次、内存分布、常量和API总览，主要展现本系统API在各个模块中的分布。

2.1 源文件目录结构

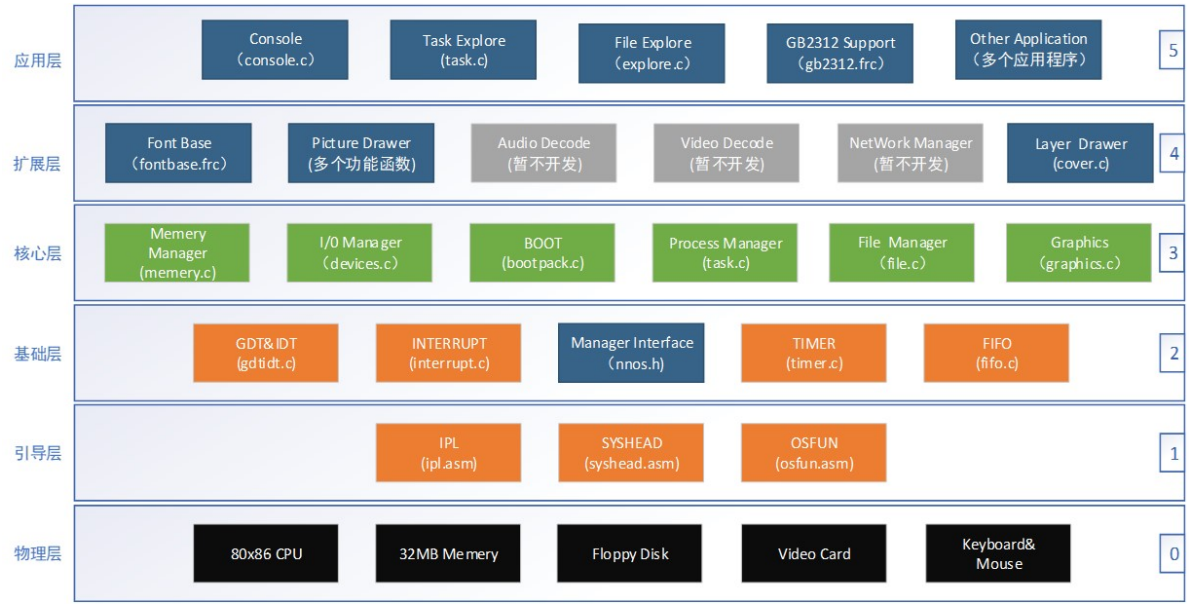
> 请参考[NNOS参考手册]。

2.2 系统内存分布

名称	地址范围	大小	用途	备注
引导区	0x00000000 ~ 0x000fffff	1MB	保存BIOS和VRAM等内容	多次调用，后变为空
数据区	0x00100000 ~ 0x00267fff	1440KB	保存软盘内容	\
保留区	0x00268000 ~ 0x0026f7ff	32KB	无数据，预留备用	\
IDT	0x0026f800 ~ 0x0026ffff	2KB	保存中断描述表	\
GDT	0x00270000 ~ 0x0027ffff	64KB	保存全局描述表	\
系统区	0x00280000 ~ 0x002fffff	512KB	保存bootpack	系统主程序
系统区	0x00300000 ~ 0x003fffff	1MB	保存栈等内容	\
剩余区	0x00400000以后	\	空白区域	可供用户使用

2.3 各层API分布

> 系统层次框图



三、引导层API详解

> 本层API的声明，全部在`nnos.h`中。
>
> `nnos.h`为系统核心库的头文件，包含全部引导层和基础层API。本节仅解析`osfun`的部分；
>
> `nnos.h`引用两个核心库，`sysdef.h`（系统常量表）和`systructual.h`（系统结构体库）。
>
> 【注意】本层全部API皆为系统级，仅可出现在应用层（不含应用层）以下，否则将出现安全问题和不可预知的错误。

3.1 ipl.asm

> 本模块无接口。详见源文件注释。
>
> 所在层：引导层

3.2 syshead.asm

> 本模块无接口。详见源文件注释。
>
> 所在层：

3.3 osfun.asm

> 本模块为系统底层接口实现。这些API大都直接操作硬件或与硬件密切相关。
>
> 【注意】本节（其他章节同）不是对函数实现的详解，而是对函数名、参数、返回值及其功能的详细解释。
>
> *如需了解函数的具体实现，请直接查看相应源文件，在源文件中有详细注释*

3.3.1 总览

函数名	功能	参数	返回值	备注
~~io_hlt~~	CUP停机	无	空	osfun:_io_hlt()
io_cli	关中断	无	空	osfun:_io_cli()
io_sti	开中断	无	空	osfun:_io_sti()
io_stihlt	开中断并停机	无	空	osfun:_io_stihlt()
io_in8	从I/0端口读8位数据，指明端口号	(int)**port**	int	int io_in8(int port)
io_out8	向I/0端口8位写数据，指明端口号和数据	(int)**port**, (int)**data**	空	void io_in8(int port,int data)
io_in16	从I/0端口16位读数据，指明端口号	(int)**port**	int	尚未实现
io_out16	向I/0端口写16位数据，指明端口号	(int)**port**, (int)**data**	空	尚未实现
io_in32	从I/0端口读数据，32位，指明端口号	(int)**port**	int	尚未实现
io_out32	向I/0端口写数据，32位，指明端口号	(int)**port**, (int)**data**	空	尚未实现
io_load_eflags	加载eflags信息	(int)**eflags**	int	int load_eflags(int eflags)
io_store_eflags	保存信息到eflags	无	空	void io_store_eflags()
load_gdtr	加载gdtr	(int)**limit**, (int)**addr**	空	void load_gdtr(int limit,int addr)
asm_inthandler21	响应键盘中断，保护现场、调用键盘中断处理器、恢复现场	空	空	void asm_inthandler21()
asm_inthandler27	响应某些机器开机自用生成的中断，避免系统启动失败	空	空	void asm_inthandler27()

函数名	功能	参数	返回值	备注
asm_inthandler2c	响应鼠标中断，保护现场、调用鼠标中断处理器、恢复现场	空	空	void asm_inthandler2c()
asm_inthandler20	响应定时器中断，保护现场、调用定时器中断处理器、恢复现场	空	空	void asm_inthandler20()
asm_inthandler0d	应用程序一般异常中断处理函数	空	空	void asm_inthandler0d();
asm_inthandler0c	应用程序栈异常中断处理函数	空	空	void asm_inthandler0c();
asm_inthandler00	除法异常，DIV、IDIV指令异常处理函数	空	空	void asm_inthandler00()
asm_inthandler01	调试异常处理函数，适用于所有指令	空	空	void asm_inthandler01()
asm_inthandler03	单字节陷阱异常处理函数	空	空	void asm_inthandler03()
asm_inthandler04	溢出异常处理函数	空	空	void asm_inthandler04()
asm_inthandler0a	无效TSS异常处理函数	空	空	void asm_inthandler0a()
asm_inthandler0b	段不存在异常处理函数	空	空	void asm_inthandler0b()
asm_inthandler05	边界检查异常处理函数	空	空	void asm_inthandler05()
asm_inthandler08	双重故障异常	空	空	void asm_inthandler08()
asm_sys_api	系统API中断处理函数	空	空	void asm_sys_api()
load_cr0	加载cr0信息	空	int	int load_cr0()
store_cr0	保存cr0信息	(int)**cr0**	空	void store_cr0(int cr0)

函数名	功能	参数	返回值	备注
memtest_sub	内存检查子函数的汇编语言实现，绕过C语言编译器的优化	(unsigned int)**start**,(unsigned int)**end**	unsigned int	unsigned int memtest_sub(unsigned int start,unsigned int end)
mts_loop	/	/	/	osfun:mts_loop
mts_fin	/	/	/	osfun:mts_fin

3.3.2 ~~io_hlt()~~

- 函数声明：`void io_hlt();`
- 功能：CUP停机等待
- 参数：无参
- 返回值：空
- 调用示例：

```
while(1){
    //若32位通用缓冲区为空，则CPU停机
    if(getFIFOBufferStatus32(&FIFOBuffer32) == 0){
        //CPU停机
        io_hlt();
    }
    else{
        //运行
    }
}
```

> **注意**：
>
> 在类似示例场景中，**不推荐**使用`io_hlt()`，请使用`io_stihlt()`。
>
> 原则上，非测试阶段系统中不会出现`io_hlt()`和`io_stihlt()`，这两个函数会使CPU停机。

3.3.3 io_cli()

- 函数声明：`void io_cli();`
- 功能：关中断。
- 参数：无参
- 返回值：空
- 调用示例：

```
while(1){
  //按F4强制结束应用程序
    if (data == 0x3e + 256 && keyShift != 0 && keyWin != 0){
        task = keyWin->task;
        if (task != 0 && task->tss.ss0 != 0){   //F4强制结束应用程序，应用运行是ss0必非0
            io_cli();   //关中断，禁止任务切换
            task->tss.eax = (int) &(task->tss.esp0);
            task->tss.eip = (int) asm_end_app;
            sysprintl(task->console,"\n[F4]Forced kill the App.\n");
            io_sti(); //开中断
        }
    }
}
```

> 说明：
>
> 本示例中，在结束应用前调用`io_cli()`关中断，目的是暂时禁止进程管理模块或其他操作在程序强制关闭时切换任务，造成关闭失败或切换到一个已经被关闭的进程（空指针）。保证强制结束进程的原子性。
>
> 实际上，在任何需要保证操作原子性的程序断中，你都可以使用`io_cli()`和`io_sti()`包围该程序段首尾。
>
> 该操作会关闭系统中断，**请谨慎使用**，并且仅在必要时（如原子操作）使用。原则上，在出现`io_cli()`的程序段后方，**必须**有一个`io_sti()`。以保证其他需要中断的系统模块正常运行。

3.3.4 io_sti()

- 函数声明：`void io_sti();`
- 功能：开中断。
- 参数：无参
- 返回值：空
- 调用示例：

```
while(1){
    io_cli();
    if(getFIFOBufferStatus32(&FIFOBuffer32) == 0){ //若32位通用缓冲区为空，则开中断
        taskSleep(taskA); //任务休眠
        io_sti(); //任务休眠后再开中断
    }
}
```

> 该函数常与`io_cti()`配合使用，包围一个不允许中断的原子操作，置于该原子操作末尾；
>
> 原则上，任何`io_cli()`后面有限的操作之后**必须**有一个`io_sti()`重新开中断。

3.3.5 io_stihlt()

- 函数声明：`void io_stihlt();`
- 功能：开中断并停机。
- 参数：无参
- 返回值：空
- 调用示例：

```
while(1){
    io_cli();
    if(getFIFOBufferStatus32(&FIFOBuffer32) == 0){ //若32位通用缓冲区为空，则开中断
        io_stihlt(); //先执行开中断再执行停机等待指令，但两条指令不能分开先后执行，否则其间有中断发生会导致崩溃
    }
}
```

> 由于`io_hlt()`会致使系统断流，故仅供测试时使用；
>
> 本函数可看作可用的`io_hlt()`；
>
> 本函数在CPU停机等待前先开中断，以保证新的中断发生时CPU可被唤醒；
>
> 本函数为原子操作，**不能**用*先调用`io_sti()`再调用`io_hlt()`*代替，因为在后两者之间可能会发生中断，导致系统崩溃。

3.3.6 io_in8()

- 函数声明：`int io_in8(int port);`
- 功能：从端口号为`port`的I/O端口读入数据，8位，指明端口号。
- 参数：端口号(port)
- 返回值：从`port`指明的端口读到的数据，int类型
- 调用示例：

```
void waitKBCReady(){ //CPU快，键盘慢，需要等待（此代码应改进为响应）
    while(1){
        if((io_in8(PORT_KEYSTA) & KEYSTA_NOTREADY) == 0){ //读取键盘状态，返回值与“键盘未准备好”代码作逐位与，为0则停止等待（确认数据的到倒数第二位为0，从低位开始数第二位）
            break; //停止等待，跳出循环
        }
    }
}
```

3.3.7 io_out8()

- 函数声明：`void io_in8(int port,int data);`
- 功能：向端口号为`port`的I/O端口输出数据或指令，8位，指明端口号。
- 参数：端口号(port)、待输出的数据(data)
- 返回值：空
- 调用示例：

```
io_out8(PIC1_IMR,0xef); //允许接收鼠标中断（(11101111)

io_out8(PORT_KEYCMD,KEYCMD_WRITE_MODE); //将键盘控制电路模式指令0x60写入键盘控制电路端口的寄存器

io_out8(PORT_KEYDAT,KBC_MODE_MOUSE); //将键盘控制电路模式设置为0x47，鼠标模式
```

> 本文的所涉及函数调用示例之常量参数及其简要说明，皆可在源文件：`sysdef.h`中找到。下文也将详细介绍这些常量。

```
#define PIC1_IMR 0x00a1
#define PORT_KEYCMD 0x0064          //键盘控制电路端口号
#define KEYCMD_WRITE_MODE 0x60      //键盘控制电路模式设置指令
#define PORT_KEYDAT 0x0060          //键盘设备端口号
#define KBC_MODE_MOUSE 0x47         //键盘控制电路模式之鼠标模式
```

3.3.8 *io_in16()*

- 函数声明：`int io_in16(int port);`
- 功能：从端口号为`port`的I/O端口读入数据，16位，指明端口号。
- 参数：端口号(port)
- 返回值：从`port`指明的端口读到的数据，int类型
- 调用示例：

```
//暂无（目前系统中尚未实现或使用过此API）
```

3.3.9 *io_out16()*

- 函数声明：`void io_in16(int port,int data);`
- 功能：向端口号为port的I/O端口写入数据或指令，16位，指明端口号。
- 参数：端口号(port)、待输出的数据(data)
- 返回值：空
- 调用示例：

```
//暂无（目前系统中尚未实现或使用过此API）
```

3.3.10 *io_in32()(*

- 函数声明：`int io_in32(int port);`
- 功能：从端口号为port的I/O端口读入数据，32位，指明端口号
- 参数：端口号(port)
- 返回值：从`port`指明的端口读到的数据，`int`类型
- 调用示例：

```
//暂无（目前系统中尚未实现或使用过此API）
```

3.3.11 *io_out32()*

- 函数声明：`void io_in32(int port,int data);`
- 功能：向端口号为`port`的I/O端口输出数据或指令，32位，指明端口号。
- 参数：端口号(port)、待输出的数据(data)。
- 返回值：空
- 调用示例：

```
//暂无（目前系统中尚未实现或使用过此API）
```

3.3.12 io_load_eflags()

- 函数声明：`void io_store_eflags;`
- 功能：加载EFLAGS信息。
- 参数：无参
- 返回值：从标志寄存器读取到的状态信息，`int`类型
- 调用示例：

```
/*画板设定*/
void setPalette(int start,int end,unsigned char *rgb){  //解析table_rgb，将颜色信息写入到显存
    int i,eflags;
    eflags = io_load_eflags();  //记录中断许可标志值
    io_cli();   //屏蔽中断
    io_out8(0x03c8,start); //向显卡备写入画板号码，通过IN指令从设备取得电气信号，区分不同设备，使用不同的端口号(port)，0x03c8为显卡画板端口号
    for(i = start;i <= end;i++){ //循环，从头到尾遍历画板
        io_out8(0x03c9,rgb[0]/4); //向0x03c9写入RGB颜色编码
        io_out8(0x03c9,rgb[1]/4);
        io_out8(0x03c9,rgb[2]/4);
        rgb +=3; //数组向后偏移3位（3个元素一组形成RGB编码）
    }
    io_store_eflags(eflags); //保存EGLAGS标志寄存器信息
}
```

> CPU中有一个由16位flag寄存器扩充而来的32位标志寄存器`EFLAGS`，用作进位、中断等。
>
> 本函数通常与`io_store_eflags()`配合使用，原则上，每次调用本函数保存或更新标志寄存器状态前，都应先读取寄存器状态。
>
> **注意** 中断指令无`JN`、`JNC`,只能通过读取`EFLAGS`,检查第`9`位（中断标志位）的值，进而判断

3.3.13 io_store_eflags()

- 函数声明：`void io_store_eflags(int eflags);`
- 功能：保存或写入信息到`EFLAGS`寄存器。
- 参数：待写入的标志信息，`int`类型
- 返回值：空
- 调用示例：

```
/*内存检查*/
//486及以上CPU有Cache，需要先禁用Cache才能完成内存检查
unsigned int memeryCheck(unsigned int start,unsigned int end){
    char flag486 = 0; //内存检查前的准备工作
    unsigned int eflag,cr0,i;
    eflag = io_load_eflags(); //读取EFLAGS寄存器内容到elag变量
    eflag |= EFLAGS_AC_BITS;   //将eflag的值与AC位存在校验码EFLAGS_AC_BIT做或运算赋值给eflag变量，使得AC位变为1。AC位在第18位。
    io_store_eflags(eflag);   //将eflag的值保存到EFLAGS寄存器
    eflag = io_load_eflags(); //读取EFLAGS寄存器内容到elag变量（386会自动把AC位置回0）。
    if((eflag & EFLAGS_AC_BITS) != 0){ //若相同，AC位未变回0，则为486
        flag486 = 1;
    }
    eflag &= ~EFLAGS_AC_BITS; //恢复AC位数值为0
    io_store_eflags(eflag);  //保存
```

```
15
16      if(flag486 != 0){
17          cr0 = load_cr0();//读取cr0寄存器内容到cr0变量
18          cr0 |= CR0_CACHE_DISABLE;
19          store_cr0(cr0);
20      }
21
22      i = memtest_sub(start,end); //检查内存，memtest_sub()函数见osfun.asm
23
24      if(flag486 != 0){
25          cr0 = load_cr0();
26          cr0 &= ~CR0_CACHE_DISABLE; //启用Cache
27          store_cr0(cr0);
28      }
29      return i;
30  }
```

> 本函数读取中断前保存的标志位信息，恢复现场。
>
> 本函数通常与`io_load_eflags()`配合使用，原则上，每次调用本函数保存或更新标志寄存器状态前，都应先读取寄存器状态。
>
> *提示*本函数所举示例较为复杂，请忽略具体细节，仅理解本函数之用法即可。

3.3.14 load_gdtr()

- 函数声明：`void load_gdtr(int limit,int addr);`
- 功能：加载GDT地址到`GDTR`段寄存器。
- 参数：GDT地址上限(limit)、GDT基址(addr)
- 返回值：空
- 调用示例：

```
1  for(i = 0;i <= IDT_LIMIT / 8;i++){ //IDT初始化
2          setGateDesc(idt + i,0,0,0);
3      }
4      load_idtr(IDT_LIMIT,IDT_ADR); //装载到地址到全局描述符寄存器，此函数在osfun.asm
   中，使用汇编语言完成
5      setGateDesc(idt + 0x20,(int)asm_inthandler20,2 * 8,INTGATE32_PRE); //注册
   定时器中断处理函数
6      setGateDesc(idt + 0x21,(int)asm_inthandler21,2 * 8,INTGATE32_PRE); //将保
   护现场函数注册到IDT中，发生中断，CPU将自动调用asm_inthandler21
7      setGateDesc(idt + 0x27,(int)asm_inthandler27,2 * 8,INTGATE32_PRE); //2*8
   为asm_inthandler*所属的段，段号为2,乘以8左移3位，低3位另有它有用，须为0
8      setGateDesc(idt + 0x2c,(int)asm_inthandler2c,2 * 8,INTGATE32_PRE);
   //INTGATE32_PRE使中断处理有效
```

```
1  #define GDT_ADR 0x00270000      //GDT基址
2  #define GDT_LIMIT 0x0000ffff    //GDT上限
```

3.3.15 logd_idtr()

- 函数声明：`void load_idtr(int limit,int addr);`
- 功能：加载IDT地址到`IDTR`段寄存器。
- 参数：IDT地址上限(limit)、IDT基址(addr)
- 返回值：空
- 调用示例：

```
//见3.3.14load_gdtr()之调用示例
```

3.3.16 asm_inthandler21()

- 函数声明：`void asm_inthandler21();`
- 功能：响应键盘中断，保护现场、调用键盘中断处理器、恢复现场。
- 参数：空
- 返回值：空
- 调用示例：

```
/*GDT和IDT初始化函数*/
void initGdtIdt(){
    int i;
    SEGMENT_DESCRIPTOR *gdt = (struct SEGMENT_DESCRIPTOR *) GDT_ADR; //定义GDT，指定基址。GDT范围为0x270000~0x27ffff（此范围可自定义）
    GATE_DESCRIPTOR *idt = (struct GATE_DESCRIPTOR *) IDT_ADR;  //定义IDT，指定基址。IDT范围为0x26f800~0x26ffff（此范围可自定义）。

    for(i = 0;i <= GDT_LIMIT / 8;i++){ //GDT初始化，16位段寄存器，低3位不可用，共可表示8192个段号，即可定义8192个段号（0~8191）
        setSegmDesc(gdt + i,0,0,0); //初始化所有段，从0开始，每次加1知道8191，8字节结构体+1，每次相当于地址加8。上限0，基址0，权限0。
    }
    setSegmDesc(gdt + 1,FULL_ADR,ZERO_ADR,DATE32_RW_PRE); //设置段1，上限0xffffffff（4GB），基址0，权限0x4092
    setSegmDesc(gdt + 2,BOOT_LIMIT,BOOT_ADR,CODE32_ER_PRE); //设置段2，上限0x0007ffff（512KB）,基址,0x00280000,权限0x409a,存放bootpack
    load_gdtr(GDT_LIMIT,GDT_ADR); //段2涵盖了整个bootpack

    for(i = 0;i <= IDT_LIMIT / 8;i++){ //IDT初始化
        setGateDesc(idt + i,0,0,0);
    }
    load_idtr(IDT_LIMIT,IDT_ADR); //装载到地址到全局描述符寄存器，此函数在osfun.asm中，使用汇编语言完成
    setGateDesc(idt + 0x20,(int)asm_inthandler20,2 * 8,INTGATE32_PRE); //注册定时器中断处理函数
    setGateDesc(idt + 0x21,(int)asm_inthandler21,2 * 8,INTGATE32_PRE); //将保护现场函数注册到IDT中，发生中断，CPU将自动调用asm_inthandler21
    setGateDesc(idt + 0x27,(int)asm_inthandler27,2 * 8,INTGATE32_PRE); //2*8为asm_inthandler*所属的段，段号为2,乘以8左移3位，低3位另有它有用，须为0
    setGateDesc(idt + 0x2c,(int)asm_inthandler2c,2 * 8,INTGATE32_PRE); //INTGATE32_PRE使中断处理有效
    //setGateDesc(idt + 0x40,(int)asm_sysprint,2*8,INTGATE32_PRE); //注册用于字符输出API的中断处理函数
    //setGateDesc(idt + 0x40,(int)asm_sys_api,2*8,INTGATE32_PRE); //注册用于处理系统API中断的函数
    setGateDesc(idt + 0x40,(int)asm_sys_api,2*8,INTGATE32_PRE + 0x60); //注册用于处理系统API中断的函数，权限偏移标识应用程序
    setGateDesc(idt + 0x0d,(int)asm_inthandler0d,2 * 8,INTGATE32_PRE); //应用程序一般异常中断处理函数
    setGateDesc(idt + 0x0c,(int)asm_inthandler0c,2 * 8,INTGATE32_PRE); //应用程序栈异常中断处理函数
    setGateDesc(idt + 0x00,(int)asm_inthandler00,2 * 8,INTGATE32_PRE); //除法异常，DIV、IDIV指令异常
    setGateDesc(idt + 0x01,(int)asm_inthandler01,2 * 8,INTGATE32_PRE); //调试异常，所有指令
```

```
    setGateDesc(idt + 0x03,(int)asm_inthandler03,2 * 8,INTGATE32_PRE); //单字节陷阱异常
    setGateDesc(idt + 0x04,(int)asm_inthandler04,2 * 8,INTGATE32_PRE); //溢出异常
    setGateDesc(idt + 0x0a,(int)asm_inthandler0a,2 * 8,INTGATE32_PRE); //无效TSS异常
    setGateDesc(idt + 0x0b,(int)asm_inthandler0b,2 * 8,INTGATE32_PRE); //段不存在异常
    setGateDesc(idt + 0x05,(int)asm_inthandler05,2 * 8,INTGATE32_PRE); //边界检查异常
    setGateDesc(idt + 0x08,(int)asm_inthandler08,2 * 8,INTGATE32_PRE); //双重故障异常
}
```

> IDT定义
>
> ```
> //定义IDT，指定基址。IDT范围为0x26f800~0x26ffff（此范围可自定义）。
> GATE_DESCRIPTOR *idt = (struct GATE_DESCRIPTOR *) IDT_ADR;
> ```
>
> 中断描述符设置函数setGateDesc():
>
> ```
> /*中断描述符设置*/
> void setGateDesc(GATE_DESCRIPTOR *gateDescriptor,int offset,int selector,int accessPre){
>     gateDescriptor->SELECTOR = selector;
>     gateDescriptor->LOW_OFFSET = offset & 0xffff;
>     gateDescriptor->HIGH_OFFSET = (offset >> 16) & 0xffff;
>     gateDescriptor->ACCESS_PER = accessPre & 0xff;
>     gateDescriptor->DW_COUNT = (accessPre >> 8) & 0xff;
> }
> ```
>
> 传入中断处理函数的首地址、处理函数首地址、扇区和权限控制常量。
>
> 本示例中所有常量之解释可参看系统常量详解章节。

3.3.17 asm_inthandler27()

- 函数声明：`void asm_inthandler27();`
- 功能：响应某些机器开机自用生成的中断，避免系统启动失败。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。

3.3.17 asm_inthandler2c()

- 函数声明：`void asm_inthandler2c();`
- 功能：响应鼠标中断，保护现场、调用鼠标中断处理器、恢复现场。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。

3.3.18 asm_inthandler20()

- 函数声明：`void asm_inthandler20();`
- 功能：响应定时器中断，保护现场、调用定时器中断处理器、恢复现场。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。

3.3.19 asm_inthandler0d()

- 函数声明：`void asm_inthandler0d();`
- 功能：应用程序一般异常中断处理函数。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 关于**应用程序一般异常**的详细解释，将在下一版中给出。

3.3.20 asm_inthandler0c()

- 函数声明：`void asm_inthandler0c();`
- 功能：应用程序栈异常中断处理函数。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 关于**应用程序栈异常中断**的详细解释，将在下一版中给出。

3.3.21 asm_inthandler00()

- 函数声明：`void asm_inthandler00();`
- 功能：除法异常，DIV、IDIV指令异常处理函数。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 除法异常，比如应用程序存在一个`1 / 0`的运算，将触发此异常。

3.3.22 asm_inthandler01()

- 函数声明：`void asm_inthandler01();`
- 功能：调试异常处理函数，适用于所有指令。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 关于**调试异常**的详细解释，将在下一版本中给出。

3.3.23 asm_inthandler03()

- 函数声明：`void asm_inthandler03();`
- 功能：单字节陷阱异常处理函数。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 关于**单字节陷阱异常**的详细解释，将在下一版本中给出。

3.3.24 asm_inthandler04()

- 函数声明：`void asm_inthandler04();`
- 功能：溢出异常处理函数。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 关于**溢出异常**的详细解释，将在下一版本中给出。

3.3.25 asm_inthandler0a()

- 函数声明：`void asm_inthandler0a();`
- 功能：无效TSS异常处理函数。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 关于**无效TSS异常**的详细解释，将在下一版本中给出。

3.3.26 asm_inthandler0b()

- 函数声明：`void asm_inthandler0b();`
- 功能：段不存在异常处理函数。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 关于**段不存在异常**的详细解释，将在下一版本中给出。

3.3.27 asm_inthandler05()

- 函数声明：`void asm_inthandler05();`
- 功能：边界检查异常处理函数。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 关于**边界检查异常**的详细解释，将在下一版本中给出。

3.3.28 asm_inthandler08()

- 函数声明：`void asm_inthandler08();`
- 功能：双重故障异常。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。
>
> 关于**双重故障异常**的详细解释，将在下一版本中给出。

3.3.29 asm_sys_api()

- 函数声明：`void asm_sys_api();`
- 功能：系统API中断处理函数。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看asm_inthandler21()之示例。

3.3.30 load_cr0()

- 函数声明：`int load_cr0();`
- 功能：加载cr0信息。
- 参数：空
- 返回值：int
- 调用示例：

> 该函数用途有待补充；
>
> 该函数实现的位置为`osfun.asm`；
>
> 具体实现：
>
> ```
> 1 _load_cr0:                ;int load_cr0();
> 2     MOV EAX,CR0           ;将寄存器CR0中的数据写入EAX
> 3     RET                   ;返回调用者
> ```
>
> 若您能补充相关信息，请在QQ群内讨论或者发信息至邮箱tianhehechu@qq.com。

3.3.31 store_cr0()

- 函数声明：`void store_cr0(int cr0);`
- 功能：保存cr0信息。
- 参数：cr0的地址(cr0)
- 返回值：空
- 调用示例：

> 该函数用途有待补充；
>
> 该函数实现的位置为`osfun.asm`；
>
> 具体实现：

```
_store_cr0:                     ;void store_cr0(int cr0);
    MOV EAX,[ESP+4]             ;将栈顶指针偏移4，所指数据存取EAX
    MOV CR0,EAX                 ;将EAX中的数据写入CR0。CR0不能直接赋值。
    RET                         ;返回调用者
```

若您能补充相关信息，请在QQ群内讨论或者发信息至邮箱tianhehechu@qq.com。

3.3.32 memtest_sub()

- 函数声明：`unsigned int memtest_sub(unsigned int start,unsigned int end);`
- 功能：内存检查子函数，检查内存是否存在致命错误，统计内存信息。
- 参数：内存首址(start)，内存尾址(end)
- 返回值：内存大小，unsigned int类型
- 调用示例：

```
/*内存检查*/
//486及以上CPU有Cache，需要先禁用Cache才能完成内存检查
unsigned int memeryCheck(unsigned int start,unsigned int end){
    char flag486 = 0; //内存检查前的准备工作
    unsigned int eflag,cr0,i;
    eflag = io_load_eflags(); //读取EFLAGS寄存器内容到elag变量
    eflag |= EFLAGS_AC_BITS;   //将eflag的值与AC位存在校验码EFLAGS_AC_BIT做或运算赋值给eflag变量，使得AC位变为1。AC位在第18位。
    io_store_eflags(eflag);   //将eflag的值保存到EFLAGS寄存器
    eflag = io_load_eflags(); //读取EFLAGS寄存器内容到elag变量（386会自动把AC位置回0）。
    if((eflag & EFLAGS_AC_BITS) != 0){ //若相同，AC位未变回0，则为486
        flag486 = 1;
    }
    eflag &= ~EFLAGS_AC_BITS; //恢复AC位数值为0
    io_store_eflags(eflag);  //保存

    if(flag486 != 0){
        cr0 = load_cr0();//读取cr0寄存器内容到cr0变量
        cr0 |= CR0_CACHE_DISABLE;
        store_cr0(cr0);
    }

    i = memtest_sub(start,end); //检查内存，memtest_sub()函数见osfun.asm

    if(flag486 != 0){
        cr0 = load_cr0();
        cr0 &= ~CR0_CACHE_DISABLE; //启用Cache
        store_cr0(cr0);
    }
    return i;
}
```

该子函数若使用c语言实现，将出现无法解决的错误，故只能使用汇编语言实现。有关此错误的详细信息，将在下一版本中给出。

3.3.33 mts_loop()

- 函数声明：无
- 功能：待补充。
- 参数：待补充。
- 返回值：待补充
- 调用示例：无

> 此函数缺乏具体信息。位置为：`osfun.asm`。
>
> 具体实现：

```
mts_loop:
    MOV EBX,EAX
    ADD EBX,0xffc               ;p=i+0xff0;
    MOV EDX,[EBX]               ;Old=*p;
    MOV [EBX],ESI               ;*p=pat0
    XOR DWORD [EBX],0xffffffff  ;*p^=0xffffffff;
    CMP EDI,[EBX]               ;if(*p!=pat1)goto fin;
    JNE mts_fin
    XOR DWORD [EBX],0xffffffff  ;*^=0xffffffff
    CMP ESI,[EBX]               ;if(*p!=pat0)goto fin;
    JNE mts_fin
    MOV [EBX],EDX               ;*p=old
    ADD EAX,0x1000              ;i+=0x1000
    CMP EAX,[ESP+12+8]          ;if(i<=end) goto mts_loop;
    JBE mts_loop
    POP EBX
    POP ESI
    POP EDI
    RET
```

> 若您能补充相关信息，请在QQ群内讨论或者发信息至邮箱tianhehechu@qq.com。

3.3.34 mts_fin()

- 函数声明：无
- 功能：待补充。
- 参数：待补充。
- 返回值：待补充
- 调用示例：无

> 此函数缺乏具体信息。位置为：`osfun.asm`。
>
> 具体实现：

```
mts_fin:
    MOV [EBX],EDX
    POP EBX
    POP ESI
    POP EDI
    RET
```

> 若您能补充相关信息，请在QQ群内讨论或者发信息至邮箱tianhehechu@qq.com。

四、基础层API详解

> 本层API的声明，全部在`nnos.h`中。
>
> `nnos.h`为系统核心库的头文件，包含全部引导层和基础层API。对于osfun的部分，本节不再重复解析；
>
> `nnos.h`引用两个核心库，`sysdef.h`（系统常量表）和`systructual.h`（系统结构体库）。
>
> 【注意】本层全部API皆为系统级，仅可出现在应用层（不含应用层）以下，否则将出现安全问题和不可预知的错误。

4.1 gdt.c的API

4.1.1 总览

函数名	功能	参数	返回值	备注
initGdtIdt	初始化GDT和IDT	空	空	
setSegmDesc	设定GDT（全局描述符）	(SEGMENT_DESCRIPTOR *)**segmentDescriptor**, (unsigned int)**limit**,(int)**base**, (int)**accessPre**	空	SEGMENT_DESCRIPTOR为结构体，见`systructual.h`
setGateDesc	设定IDT（中断描述符）	(SHEET *)**sht**,(char)**act**	空	

4.1.2 initGdtIdt()

- 函数声明：`void initGdtIdt();`
- 功能：初始化GDT和IDT。
- 参数：空
- 返回值：空
- 调用示例：

```
void NNOSMain(){
    //省略前驱操作
    initGdtIdt();   //初始化GDT（全局描述符）和IDT（中断描述符）
    initPIC();      //初始化PIC(可编程中断处理器)
    io_sti(); //允许中断发生（初始化PIC时禁止了中断发生），函数见osfun.asm
    initFIFOBuffer32(&FIFOBuffer32,buffer,128,0); //初始化32位通用缓冲区
    *((int *) 0x0fec) = (int)&FIFOBuffer32; //存储缓冲区首地址
    initPIT();      //初始化PIT(可编程间隔定时器)
    //省略后继操作
}
```

4.1.3 setSegmDesc()

- 函数声明：`void setSegmDesc(SEGMENT_DESCRIPTOR *segmentDescriptor,unsigned int limit,int base,int accessPre);`
- 功能：设定GDT（全局描述符）。
- 参数：全局描述符段的地址(segmentDescriptor)、范围(limit)、基址(base)和权限(accessPre)
- 返回值：空
- 调用示例：

```
/*GDT和IDT初始化函数*/
void initGdtIdt(){
    int i;
    SEGMENT_DESCRIPTOR *gdt = (struct SEGMENT_DESCRIPTOR *) GDT_ADR; //定义GDT，指定基址。GDT范围为0x270000~0x27ffff（此范围可自定义）
    GATE_DESCRIPTOR *idt = (struct GATE_DESCRIPTOR *) IDT_ADR;  //定义IDT，指定基址。IDT范围为0x26f800~0x26ffff（此范围可自定义）。

    for(i = 0;i <= GDT_LIMIT / 8;i++){ //GDT初始化，16位段寄存器，低3位不可用，共可表示8192个段号，即可定义8192个段号（0~8191）
        setSegmDesc(gdt + i,0,0,0); //初始化所有段，从0开始，每次加1知道8191，8字节结构体+1，每次相当于地址加8。上限0，基址0，权限0。
    }
    setSegmDesc(gdt + 1,FULL_ADR,ZERO_ADR,DATE32_RW_PRE); //设置段1，上限0xffffffff（4GB），基址0，权限0x4092
    setSegmDesc(gdt + 2,BOOT_LIMIT,BOOT_ADR,CODE32_ER_PRE); //设置段2，上限0x0007ffff（512KB）,基址,0x00280000,权限0x409a,存放bootpack
    load_gdtr(GDT_LIMIT,GDT_ADR); //段2涵盖了整个bootpack

    for(i = 0;i <= IDT_LIMIT / 8;i++){ //IDT初始化
        setGateDesc(idt + i,0,0,0);
    }
    load_idtr(IDT_LIMIT,IDT_ADR); //装载到地址到全局描述符寄存器，此函数在osfun.asm中，使用汇编语言完成
    setGateDesc(idt + 0x20,(int)asm_inthandler20,2 * 8,INTGATE32_PRE); //注册定时器中断处理函数
    setGateDesc(idt + 0x21,(int)asm_inthandler21,2 * 8,INTGATE32_PRE); //将保护现场函数注册到IDT中，发生中断，CPU将自动调用asm_inthandler21
    setGateDesc(idt + 0x27,(int)asm_inthandler27,2 * 8,INTGATE32_PRE); //2*8为asm_inthandler*所属的段，段号为2,乘以8左移3位，低3位另有它有用，须为0
    setGateDesc(idt + 0x2c,(int)asm_inthandler2c,2 * 8,INTGATE32_PRE); //INTGATE32_PRE使中断处理有效
    //setGateDesc(idt + 0x40,(int)asm_sysprint,2*8,INTGATE32_PRE); //注册用于字符输出API的中断处理函数
    //setGateDesc(idt + 0x40,(int)asm_sys_api,2*8,INTGATE32_PRE); //注册用于处理系统API中断的函数
    setGateDesc(idt + 0x40,(int)asm_sys_api,2*8,INTGATE32_PRE + 0x60); //注册用于处理系统API中断的函数，权限偏移标识应用程序
    setGateDesc(idt + 0x0d,(int)asm_inthandler0d,2 * 8,INTGATE32_PRE); //应用程序一般异常中断处理函数
    setGateDesc(idt + 0x0c,(int)asm_inthandler0c,2 * 8,INTGATE32_PRE); //应用程序栈异常中断处理函数
    setGateDesc(idt + 0x00,(int)asm_inthandler00,2 * 8,INTGATE32_PRE); //除法异常，DIV、IDIV指令异常
    setGateDesc(idt + 0x01,(int)asm_inthandler01,2 * 8,INTGATE32_PRE); //调试异常，所有指令
    setGateDesc(idt + 0x03,(int)asm_inthandler03,2 * 8,INTGATE32_PRE); //单字节陷阱异常
```

```
30      setGateDesc(idt + 0x04,(int)asm_inthandler04,2 * 8,INTGATE32_PRE); //溢出异常
31      setGateDesc(idt + 0x0a,(int)asm_inthandler0a,2 * 8,INTGATE32_PRE); //无效TSS异常
32      setGateDesc(idt + 0x0b,(int)asm_inthandler0b,2 * 8,INTGATE32_PRE); //段不存在异常
33      setGateDesc(idt + 0x05,(int)asm_inthandler05,2 * 8,INTGATE32_PRE); //边界检查异常
34      setGateDesc(idt + 0x08,(int)asm_inthandler08,2 * 8,INTGATE32_PRE); //双重故障异常
35  }
```

4.1.4 setSegmDesc()

- 函数声明：`void setSegmDesc(SEGMENT_DESCRIPTOR *segmentDescriptor,unsigned int limit,int base,int accessPre);`
- 功能：设定GDT（全局描述符）。
- 参数：全局描述符段的地址(segmentDescriptor)、范围(limit)、基址(base)和权限(accessPre)
- 返回值：空
- 调用示例：

> 请参看setSegmDesc()之示例。

4.2 interrupt.c的API

4.2.1 总览

函数名	功能	参数	返回值	备注
initPIC	初始化PIC	空	空	
initPIT	初始化PIT(可编程间隔定时器)	空	空	
inthandler21	PS/2键盘中断处理器	(int *)**esp**	空	
inthandler2c	PS/2鼠标中断处理器	(int *)**esp**	空	
inthandler27	针对PIC0的不完全中断补丁	(int *)**esp**	空	
inthandler20	处理IRQ0中断，设置定时器	(int *)**esp**	空	
inthandler0c	栈异常中断处理	(int *)**esp**	int	
inthandler0d	一般异常保护	(int *)**esp**	int	
inthandler00	栈异常中断处理	(int *)**esp**	int	
inthandler01	栈异常中断处理	(int *)**esp**	int	
inthandler03	栈异常中断处理	(int *)**esp**	int	
inthandler04	栈异常中断处理	(int *)**esp**	int	
inthandler0a	无效TSS异常中断处理	(int *)**esp**	int	
inthandler0b	段不存在异常中断处理	(int *)**esp**	int	
inthandler05	段不存在异常中断处理	(int *)**esp**	int	
inthandler08	段不存在异常中断处理	(int *)**esp**	int	

4.2.2 initPIC()

- 函数声明：`void initPIC();`
- 功能：初始化GDT和IDT。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看initGdtIdt()之示例。

4.2.3 initPIT()

- 函数声明：`void initPIT();`
- 功能：初始化PIT(可编程间隔定时器)。
- 参数：空
- 返回值：空
- 调用示例：

> 请参看initGdtIdt()之示例。

4.2.4 inthandler21()

- 函数声明：`void inthandler21(int *esp);`
- 功能：PS/2键盘中断处理器。
- 参数：栈指针(esp)
- 返回值：空
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.5 inthandler2c()

- 函数声明：`void inthandler2c(int *esp);`
- 功能：PS/2鼠标中断处理器。
- 参数：栈指针(esp)
- 返回值：空
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.6 inthandler27()

- 函数声明：`void inthandler27(int *esp);`
- 功能：针对PIC0的不完全中断补丁。
- 参数：栈指针(esp)
- 返回值：空
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.7 inthandler20()

- 函数声明：`void inthandler20(int *esp);`
- 功能：处理IRQ0中断，设置定时器。
- 参数：栈指针(esp)
- 返回值：空
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.8 inthandler0c()

- 函数声明：`int inthandler0c(int *esp);`
- 功能：栈异常中断处理。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.9 inthandler0d()

- 函数声明：`int inthandler0d(int *esp);`
- 功能：一般异常保护。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.10 inthandler00()

- 函数声明：`int inthandler00(int *esp);`
- 功能：栈异常中断处理。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.11 inthandler01()

- 函数声明：`int inthandler01(int *esp);`
- 功能：栈异常中断处理。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.12 inthandler03()

- 函数声明：`int inthandler03(int *esp);`
- 功能：栈异常中断处理。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.13 inthandler04()

- 函数声明：`int inthandler04(int *esp);`
- 功能：栈异常中断处理。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.14 inthandler0a()

- 函数声明：`int inthandler0a(int *esp);`
- 功能：无效TSS异常中断处理。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.15 inthandler0b()

- 函数声明：`int inthandler0b(int *esp);`
- 功能：段不存在异常中断处理。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.16 inthandler05()

- 函数声明：`int inthandler05(int *esp);`
- 功能：段不存在异常中断处理。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.2.17 inthandler08()

- 函数声明：`int inthandler08(int *esp);`
- 功能：段不存在异常中断处理。
- 参数：栈指针(esp)
- 返回值：int(强制结束应用程序处理器的栈指针)
- 调用示例：

> 中断处理器无需手动调用，只能被动触发。

4.3 fifo.c的API

4.3.1 总览

函数名	功能	参数	返回值	备注
~~initFIFOBuffer~~	初始化专用缓冲区	(FIFO_BUFFER *)**fifoBuffer**, (unsigned char *)**buffer**, (int)**bufferSize**	空	
~~putFIFOBuffer~~	向专用缓冲区写入数据	(FIFO_BUFFER *)**fifoBuffer**, (unsigned char)**data**	int	
~~getFIFOBufferr~~	从专用缓冲区读取数据	(FIFO_BUFFER *)**fifoBuffer**	int	
~~getFIFOBufferStatus~~	获取专用缓冲区数据状态	(FIFO_BUFFER *)**fifoBuffer**	int	
initFIFOBuffer32	初始化通用缓冲区	(FIFO_BUFFER32 *)**fifoBuffer**,(int *)**buffer**,(int)**bufferSize**,(TASK *)**task**	void	
putFIFOBuffer32	向通用缓冲区写入数据	(FIFO_BUFFER32 *)**fifoBuffer**, (int)**data**	int	
getFIFOBuffer32	从通用缓冲区读取数据	(FIFO_BUFFER32 *)**fifoBuffer**	int	
getFIFOBufferStatus32	获取通用缓冲区数据状态	(FIFO_BUFFER32 *)**fifoBuffer**	int	

4.3.2 ~~initFIFOBuffer()~~

- 函数声明：`void initFIFOBuffer(FIFO_BUFFER *fifoBuffer,unsigned char *buffer,int bufferSize);`
- 功能：初始化专用缓冲区（8位）。
- 参数：缓冲区地址(fifoBuffer)，缓冲区数组地址(buffer)，缓冲区大小(bufferSize)
- 返回值：空
- 调用示例：

```
void NNOSMain(){ //不能用return
    FIFO_BUFFER FIFOBuffer
    initFIFOBuffer(&FIFOBuffer,buffer,128); //初始化专用缓冲区
    putFIFOBuffer(&FIFOKeyCMD,KEYCMD_LED_STATUS); //初始化键盘操作命令缓冲区数据
}
```

4.3.3 ~~putFIFOBuffer()~~

- 函数声明：`int putFIFOBuffer(FIFO_BUFFER *fifoBuffer,unsigned char data);`
- 功能：向专用缓冲区写入数据（8位）。
- 参数：缓冲区地址(fifoBuffer)，要写入的数据(data)
- 返回值：int(正常返回0，溢出返回-1)
- 调用示例：

> 请参照initFIFOBuffer()之示例。

4.3.4 ~~getFIFOBuffer()~~

- 函数声明：`int getFIFOBuffer(FIFO_BUFFER *fifoBuffer);`
- 功能：从专用缓冲区读取数据（8位）。
- 参数：缓冲区地址(fifoBuffer)
- 返回值：int(从缓冲区读取的数据)
- 调用示例：

```
while(1){ //永真循环
    if(getFIFOBufferStatus(&FIFOBuffer) == 0){ //若缓冲区为空，则开中断
        taskSleep(taskA); //任务休眠
        io_sti(); //任务休眠后再开中断
    }
    else{ //消除外层判断（经验证不如保留，故不变）
        data = getFIFOBuffer(&FIFOBuffer);
        io_sti();
        if (keyWin != 0 && keyWin->flags == 0) {    //窗口关闭
            if (shtctl->top == 1) { //最高图层为1，只剩桌面和鼠标
                keyWin = 0;
            }
            else{
                keyWin = shtctl->sheets[shtctl->top - 1];
                keywinOn(keyWin);
            }
        }
    }
}
```

4.3.5 ~~getFIFOBufferStatus()~~

- 函数声明：`int getFIFOBufferStatus(FIFO_BUFFER *fifoBuffer);`
- 功能：获取专用缓冲区数据状态（8位）。
- 参数：缓冲区地址(fifoBuffer)
- 返回值：int(缓冲区大小)
- 调用示例：

> 请参照getFIFOBuffer()之示例。

4.3.6 initFIFOBuffer32()

- 函数声明：`void initFIFOBuffer32(FIFO_BUFFER32 *fifoBuffer,int *buffer,int bufferSize,TASK *task);`
- 功能：初始化32位缓冲区。
- 参数：缓冲区地址(fifoBuffer)，缓冲区数组地址(buffer)，缓冲区大小(bufferSize)，所属任务指针(task)
- 返回值：空

- 调用示例：

```
void NNOSMain(){ //不能用return
    FIFO_BUFFER32 FIFOBuffer32,FIFOKeyCMD;.
    initFIFOBuffer32(&FIFOBuffer32,buffer,128,0); //初始化32位通用缓冲区
    initFIFOBuffer32(&FIFOKeyCMD,keyCMDBuffer,32,0); //初始化键盘操作命令缓冲区
    putFIFOBuffer32(&FIFOKeyCMD,KEYCMD_LED_STATUS); //初始化键盘操作命令缓冲区数据
    putFIFOBuffer32(&FIFOKeyCMD,keyLEDS); //将LED等状态信息写入键盘操作命令缓冲区
}
```

4.3.7 putFIFOBuffer32()

- 函数声明：`int putFIFOBuffer32(FIFO_BUFFER32 *fifoBuffer,int data);`
- 功能：向32位缓冲区写入数据。
- 参数：缓冲区地址(fifoBuffer)，要写入的数据(data)
- 返回值：int(正常返回0，溢出返回-1)
- 调用示例：

> 请参照initFIFOBuffer32()之示例。

4.3.8 getFIFOBuffer32()

- 函数声明：`int getFIFOBuffer32(FIFO_BUFFER32 *fifoBuffer);`
- 功能：从32位缓冲区读取数据。
- 参数：缓冲区地址(fifoBuffer)
- 返回值：int(从缓冲区读取的数据)
- 调用示例：

```
while(1){ //永真循环
    if(getFIFOBufferStatus32(&FIFOBuffer32) == 0){ //若32位通用缓冲区为空，则开中断
        taskSleep(taskA); //任务休眠
        io_sti(); //任务休眠后再开中断
    }
    else{ //消除外层判断（经验证不如保留，故不变）
        data = getFIFOBuffer32(&FIFOBuffer32);
        io_sti();
        if (keyWin != 0 && keyWin->flags == 0) {    //窗口关闭
            if (shtctl->top == 1) { //最高图层为1，只剩桌面和鼠标
                keyWin = 0;
            }
            else{
                keyWin = shtctl->sheets[shtctl->top - 1];
                keywinOn(keyWin);
            }
        }
    }
}
```

4.3.9 getFIFOBufferStatus32()

- 函数声明：`int getFIFOBufferStatus32(FIFO_BUFFER32 *fifoBuffer);`
- 功能：获取32位缓冲区数据状态。
- 参数：缓冲区地址(fifoBuffer)
- 返回值：int(缓冲区大小)

- 调用示例：

> 请参照getFIFOBuffer32()之示例。.

4.4 timer.c 的API

4.4.1 总览

函数名	功能	参数	返回值	备注
initTimer	定时器初始化	(TIMER *)timer,(FIFO_BUFFER32 *)FIFOBuffer,(int)data	空	
timerAlloc	定时器内存分配	空	空	
setTimer	定时器设定	(TIMER *)timer,(unsigned int)timeout	空	
freeTimer	定时器释放	(TIMER *)timer	空	
timerCancel	定时器关闭	(TIMER *)timer	int	
timerCancelAll	关闭全部定时器	(FIFO_BUFFER32 *)fifo	空	
setTest480	设定480个测试用定时器	(FIFO_BUFFER32 *)fifo,(int)mode	空	

4.4.2 initTimer()

- 函数声明：`void initTimer(TIMER *timer,FIFO_BUFFER32 *FIFOBuffer,int data);`
- 功能：定时器初始化
- 参数：定时器(timer)，缓冲区(FIFOBuffer)，信号(data)
- 返回值：空
- 调用示例：

```
/*控制台光标闪烁*/
CONSOLE console;
console.timer = timerAlloc();initTimer(console.timer,&task->fifo,1);setTimer(console.timer,50); //初始化定时器缓冲区
while(1){
        io_cli();
        if(getFIFOBufferStatus32(&task->fifo) == 0){
            taskSleep(task);
            io_sti();
        }
        else{
            data = getFIFOBuffer32(&task->fifo);
            io_sti();
            if(data <= 1){
                if(data != 0){
                    initTimer(console.timer,&task->fifo,0);
                    if(console.cursorC >= 0){
                        console.cursorC = COL8_FFFFFE;
                    }
                }
                else{
```

```
                    initTimer(console.timer,&task->fifo,1);
                    if(console.cursorC >= 0){
                        console.cursorC = COL8_000000;
                    }
                }
                setTimer(console.timer,50);
            }
            if(data == 2){ //光标显示
                console.cursorC = COL8_FFFFFF;
            }
            if(data == 3){ //光标隐藏
                console.cursorC = - 1;
                boxDraw8(console.sheet->buf,console.sheet->bxsize,console.cursorX,console.cursorY ,console.cursorX + 7,43,COL8_000000); //绘制光标
            }
        }
}
```

4.4.3 timerAlloc()

- 函数声明：`TIMER *timerAlloc();`
- 功能：定时器申请，为定时器分配内存
- 参数：空
- 返回值：空
- 调用示例：

```
FIFO_BUFFER32 FIFOBuffer32;
initFIFOBuffer32(&FIFOBuffer32,buffer,128,0); //初始化32位通用缓冲区
*((int *) 0x0fec) = (int)&FIFOBuffer32; //存储缓冲区首地址
TIMER *timer2,*timer3; //定时器
timer2 = timerAlloc();initTimer(timer2,&FIFOBuffer32,10);setTimer(timer2,1000); //初始化定时器缓冲区
timer3 = timerAlloc();initTimer(timer3,&FIFOBuffer32,3);setTimer(timer3,300);
```

4.4.5 setTimer()

- 函数声明：`void setTimer(TIMER *timer,unsigned int timeout);`
- 功能：定时器设定
- 参数：定时器(timer)，超时时间(timerout)
- 返回值：空
- 调用示例：

> 请参看timerAlloc()之示例。

4.4.6 timerCancel()

- 函数声明：`int timerCancel(TIMER *timer);`
- 功能：定时器设定
- 参数：定时器(timer)
- 返回值：int(关闭成功返回1，失败返回0)
- 调用示例：

```
/*关闭所有运行中的定时器*/
```

```
void timerCancelAll(FIFO_BUFFER32 *fifo){
    int e,i;
    TIMER *t;
    e = io_load_eflags(); //保护现场
    io_cli(); //关中断，防止定时器状态改变
    for (i = 0;i < TIMER_MAX;i++) {
        t = &timerList.timer[i];
        if (t->flag != TIMER_FLAG_UNUSED && t->flag2 != 0 && t->fifo ==
fifo) {
            timerCancel(t);
            freeTimer(t);
        }
    }
    io_store_eflags(e); //恢复现场
    return;
}
```

4.4.7 timerCancelAll()

- 函数声明：`void timerCancelAll(FIFO_BUFFER32 *fifo);`
- 功能：关闭全部定时器
- 参数：缓冲区(fifo)
- 返回值：int(关闭成功返回1，失败返回0)
- 调用示例：无

4.4.8 setTest480()

- 函数声明：`void setTest480(FIFO_BUFFER32 *fifo,int mode);`
- 功能：设定480个测试用定时器
- 参数：缓冲区(fifo)，模式(mode)
- 返回值：空
- 调用示例：无

> 本函数仅用于系统性能测试。
>
> 模式(`mode`)用于设置是否启用这些定时器。

五、核心层API详解

> 本层API的声明，全部在`nnos.h`中。
>
> `nnos.h`为系统核心库的头文件，包含全部引导层和基础层API。对于osfun的部分，本节不再重复解析；
>
> `nnos.h`引用两个核心库，`sysdef.h`（系统常量表）和`systructual.h`（系统结构体库）。
>
> 【注意】本层全部API皆为系统级，仅可出现在应用层（不含应用层）以下，否则将出现安全问题和不可预知的错误。

5.1 memery.c的API

5.1.1 总览

函数名	功能	参数	返回值	备注
memeryCheck	内存检查	(unsigned int)**start**, (unsigned int)**end**	unsigned int	
initMemeryManager	初始化内存块表	(MEMERY_LIST *)**memeryList**	空	
~~memeryTotal~~	获取内存总剩余量	(MEMERY_LIST *)**memeryList**	unsigned int	
memeryAlloc	分配内存	(MEMERY_LIST *)**memeryList**,(unsigned int)**size**	unsigned int	
memeryFree	释放内存	(MEMERY_LIST *)**memeryList**,(unsigned int)**addr**,(unsigned int)**size**	int	
deleteBlock	删除内存表项	(MEMERY_LIST *)**memeryList**,(int)**i**	空	
addBlock	插入内存表项	（MEMERY_LIST *)**memeryList**,(int)**i**	空	
sortMemery	内存表项按内存块大小快速排序（递增）	(MEMERY_LIST *)**memeryList**,(int)low, (int)**high**	空	
sortAddr	内存表项按内存地址大小快速排序（递增）	(MEMERY_LIST *)**memeryList**,(unsigned int)**low**,(unsigned int)**high**	空	
memeryAlloc4k	内存空间分配前向上取整，最小4K(0x)，减少碎片	(MEMERY_LIST *)**memeryList**,(unsigned int)**size**	unsigned int	
memeryFree4k	内存空间释放按4K向上取整	(MEMERY_LIST *)**memeryList**,(unsigned int)**addr**,(unsigned int)**size**	unsigned int	
clearMemery	清理内存	(MEMERY_LIST *)**memeryList**,(unsigned int)**addr**,(unsigned int)**size**	空	

5.1.2 memeryCheck()

- 函数声明：`unsigned int memeryCheck(unsigned int start,unsigned int end);`
- 功能：内存检查。
- 参数：内存首址(start)，内存尾址(end)
- 返回值：unsigned int(内存大小)
- 调用示例：

```
/*初始化内存表*/
void initMemeryManager(MEMERY_LIST *memeryList){ //初始化内存表
    memeryList->total_size = memeryCheck(0x00400000,0xbfffffff); //内存总量等于内存检查返回的内存大小
    memeryList->number = 0; //从0开始计数
    memeryList->max_number = 0;
    memeryList->lost_number = 0;
    memeryList->lost_size = 0;
}
```

> 本函数调用`osfun.asm`中的内存检查子函数`memtest_sub()`，完成内存检查，返回首址至尾址的内存大小。

5.1.3 initMemeryManager()

- 函数声明：`void initMemeryManager(MEMERY_LIST *memeryList);`
- 功能：初始化内存块表。
- 参数：内存块表(memeryList)
- 返回值：空
- 调用示例：

```
void NNOSMain(){ //不能用return
    MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR; //内存表
    initMemeryManager(memeryList);  //初始化内存管理系统
}
```

> 初始化内存时，完成内存检查、获取内存大小、初始化内存块为0，初始化最大内存块数为0，初始化丢失内存块数为0，初始化丢失内存块总大小为0。

5.1.4 ~~memeryTotal()~~

- 函数声明：`unsigned int memeryTotal(MEMERY_LIST *memeryList);`
- 功能：获取内存总剩余量。
- 参数：内存块表(memeryList)
- 返回值：unsigned int(剩余内存大小)
- 调用示例：无。

> 本函数已废弃。

5.1.5 memeryAlloc()

- 函数声明：`unsigned int memeryAlloc(MEMERY_LIST *memeryList,unsigned int size);`
- 功能：分配内存。从空闲内存块中，选择一个最合适大小的内存块，并将首地址返回。
- 参数：内存块表(memeryList)，申请内存大小(size)
- 返回值：unsigned int(分配的内存大小)
- 调用示例：

```
/*4K取整方式申请内存*/
unsigned int memeryAlloc4k(MEMERY_LIST *memeryList,unsigned int size){ //内存空间分配前向上取整，最小4K(0x)，减少碎片
    unsigned int addr; //分配到的首地址
    //size = (size + 0xfff) & BLOCK_ALIGN; //以4K为最小单位向上取整。相当于if((i & 0xfff) != 0){i = (i & 0xfffff000) + 0x1000;}
    size = (size + 0xfff) & 0xfffff000; //以4K为最小单位向上取整。相当于if((i & 0xfff) != 0){i = (i & 0xfffff000) + 0x1000;}
    addr = memeryAlloc(memeryList,size);
}
```

> 本函数不推荐直接使用，内存管理API中，存在一个调用了本函数并对内存卡进行4K对齐的内存分配方法`memeryAlloc4k()`。在大多数场景下，你可以使用`memeryAlloc4k()`代替`memeryAlloc()`，因为`memeryAlloc4k()`分配的内存可以使程序更高效地运行，并减少内存碎片。

5.1.6 memeryFree()

- 函数声明：`int memeryFree(MEMERY_LIST *memeryList,unsigned int addr,unsigned int size);`
- 功能：释放内存。获得内存的系统模块或应用程序调用本函数释放自身或局部内存，这块内存在完成简单合并后，将挂在到空闲块表。
- 参数：内存块表(memeryList)，内存首址(addr)，内存大小(size)
- 返回值：int(释放的内存大小)
- 调用示例：

```
/*4K取整方式释放内存*/
unsigned int memeryFree4k(MEMERY_LIST *memeryList,unsigned int addr,unsigned int size){ //内存空间释放按4K向上取整
    int i;
    size = (size + 0xfff) & 0xfffff000; //以4K为最小单位向上取整
    i = memeryFree(memeryList,addr,size);
    return i;
}
```

> 本函数不推荐直接使用，因为相应的内存分配函数`memeryAlloc()`不推荐直接使用。
>
> 与`memeryAlloc()`的4K对齐版本`memeryAlloc4k()`类似，`memeryFree()`也有4K对齐版本`memeryFree4k()`。
>
> **【注意】**`memeryAlloc()`须与`memeryFree()`配对使用；`memeryAlloc4k()`须与`memeryFree4k()`配对使用；
>
> 亦即，若一块内存由`memeryAlloc()`**直接**申请获得，则在释放它时应当使用`memeryFree()`；若一块内存由`memeryAlloc4k()`申请获得，则释放它时应当使用`memeryFree4k()`。

5.1.7 deleteBlock()

- 函数声明：`void deleteBlock(MEMERY_LIST *memeryList,int i);`
- 功能：删除内存表项。
- 参数：内存块表(memeryList)，内存块索引(i)
- 返回值：空
- 调用示例：

```
/*内存分配*/
```

```
unsigned int memeryAlloc(MEMERY_LIST *memeryList,unsigned int size){ //内存分配
    int i; //计数器
    unsigned int addr; //分配到的首地址
    sortMemery(memeryList,0,memeryList->max_number); //内存块从大到小排序
    for(i = 0;i < memeryList->number;i++){
        if(memeryList->free[i].size >= size){
            addr = memeryList->free[i].addr;    //分配首地址为本块起始地址
            memeryList->free[i].addr += size; //本块地址偏移已分配大小
            memeryList->free[i].size -= size; //本块容量减去分配容量
            memeryList->total_free_size -= size; //总容量减去分配容量
            if(memeryList->free[i].size == 0){ //删除空块
                deleteBlock(memeryList,i);
            }
            sortMemery(memeryList,0,memeryList->number); //内存块从大到小排序
            return addr;
        }
    }
    return 0;
}
```

5.1.8 addBlock()

- 函数声明：`void deleteBlock(MEMERY_LIST *memeryList,int i);`
- 功能：插入内存表项。
- 参数：内存块表(memeryList)，插入位置索引(i)
- 返回值：空
- 调用示例：

```
if(i > 0){
        if(memeryList->free[i - 1].addr + memeryList->free[i - 1].size == addr){ //前有可用内存
            memeryList->free[i - 1].size += size;
            if(addr + size == memeryList->free[i].addr){
                memeryList->free[i - 1].size += memeryList->free[i].size;
                deleteBlock(memeryList,i);
            }
            return 0;
        }
        else{
            if(addr + size == memeryList->free[i].addr){
                memeryList->free[i].size += size;
                memeryList->free[i].addr = addr;
            }
            else{
                addBlock(memeryList,i);
                memeryList->free[i].size = size;
                memeryList->free[i].addr = addr;
                if(memeryList->max_number < memeryList->number){
                    memeryList->max_number = memeryList->number; //更新历史最大块数
                }
            }
        }
        memeryList->total_free_size += size;
        sortMemery(memeryList,0,memeryList->number);
        return 0;
```

```
    }
}
```

5.1.9 sortMemery()

- 函数声明：`void sortMemery(MEMERY_LIST *memeryList,int low,int high);`
- 功能：内存表项按内存块大小快速排序（递增）。
- 参数：内存块表(memeryList)，最低索引(low)，最高索引(high)
- 返回值：空
- 调用示例：

```
if (memeryList->number < MEMERY_LIST_SIZE) { //补丁
    for (j = memeryList->number; j > i; j--) {
        memeryList->free[j] = memeryList->free[j - 1];
    }
    memeryList->number++;
    if (memeryList->max_number < memeryList->number) {
        memeryList->max_number = memeryList->number;
    }
    memeryList->free[i].addr = addr;
    memeryList->free[i].size = size;
    memeryList->total_free_size += size;
    sortMemery(memeryList,0,memeryList->number);
    return 0;
}
```

5.1.10 sortAddr()

- 函数声明：`void sortAddr(MEMERY_LIST *memeryList,unsigned int low,unsigned int high);`
- 功能：内存表项按内存地址大小快速排序（递增）。
- 参数：内存块表(memeryList)，最低索引(low)，最高索引(high)
- 返回值：空
- 调用示例：

```
/*释放内存*/
int memeryFree(MEMERY_LIST *memeryList,unsigned int addr,unsigned int size){
//释放内存
    int i,j;
    sortAddr(memeryList,0,memeryList->number);
    for(i = 0;i < memeryList->number;i++){
        if(memeryList->free[i].addr > addr){
            break;
        }
    }

    if(i > 0){
        if(memeryList->free[i - 1].addr + memeryList->free[i - 1].size ==
addr){ //前有可用内存
            memeryList->free[i - 1].size += size;
            if(addr + size == memeryList->free[i].addr){
                memeryList->free[i - 1].size += memeryList->free[i].size;
                deleteBlock(memeryList,i);
            }
            return 0;
```

```
        }
        else{
            if(addr + size == memeryList->free[i].addr){
                memeryList->free[i].size += size;
                memeryList->free[i].addr = addr;
            }
            else{
                addBlock(memeryList,i);
                memeryList->free[i].size = size;
                memeryList->free[i].addr = addr;
                if(memeryList->max_number < memeryList->number){
                    memeryList->max_number = memeryList->number; //更新历史最大块数
                }
            }
        }
        memeryList->total_free_size += size;
        sortMemery(memeryList,0,memeryList->number);
        return 0;
    }
    //省略后继代码
    memeryList->lost_number++;
    memeryList->lost_size += size;
    sortMemery(memeryList,0,memeryList->number);
    return -1;
}
```

5.1.11 memeryAlloc4k()

- 函数声明：`unsigned int memeryAlloc4k(MEMERY_LIST *memeryList,unsigned int size);`
- 功能：内存空间分配前向上取整，最小4K(0x)，减少碎片。
- 参数：内存块表(memeryList)，申请内存大小(size)
- 返回值：unsigned int(分配内存大小)
- 调用示例：

```
unsigned char *fonts;
fonts = (unsigned char *)memeryAlloc4k(memeryList, 0x5d5d * 32); //为字库申请内存
```

> 若调用本函数申请内存，则在释放内存时应当调用`memeryFree4k()`。

5.1.12 memeryFree4k()

- 函数声明：`unsigned int memeryFree4k(MEMERY_LIST *memeryList,unsigned int addr,unsigned int size);`
- 功能：针对`memeryAlloc4k()`的内存释放函数。
- 参数：内存块表(memeryList)，内存首址(addr)，内存大小(size)
- 返回值：unsigned int(释放内存大小)
- 调用示例：

```
int *fat;
fat = (int *)memeryAlloc4k(memeryList, 4 * 2880);
memeryFree4k(memeryList, (int) fat, 4 * 2880);
```

5.1.13 *clearMemery()*

- 函数声明：`void clearMemery(MEMERY_LIST *memeryList,unsigned int addr,unsigned int size);`
- 功能：清理内存。
- 参数：内存块表(memeryList)，内存首址(addr)，内存大小(size)
- 返回值：空
- 调用示例：无。

> 本函数尚未实现。

5.2 devices.c 的API

5.26.1 总览

函数名	功能	参数	返回值	备注
~~initScreen~~	屏幕信息初始化函数	(SCREEN *)screen,(BOOT_INFO *)bootInfo	空	
waitKBCReady	等待键盘控制电路准备完毕函数	空	空	
~~initKeyboardCMD~~	初始化键盘控制电路函数	空	空	
~~enableMouse~~	激活鼠标函数	(MOUSE_CURSOR *)mouseCursor	空	
decodeMouse	鼠标解码引擎	(MOUSE_CURSOR *)mouseCursor, (unsigned char)data	int	
enableMouse32	初始化键盘控制电路函数,32位	(FIFO_BUFFER32 *)FIFOBuffer, (int)dataBase,(MOUSE_CURSOR *)mouseCursor	空	
initKeyboardCMD32	激活鼠标、键盘函数，32位	(FIFO_BUFFER32 *)FIFOBuffer, (int)dataBase	空	
sysHLT	停机等待函数	空	空	

5.2.2 ~~initScreen~~()

- 函数声明：`void initScreen(SCREEN *screen,BOOT_INFO *bootInfo);`
- 功能：屏幕信息初始化
- 参数：屏幕信息(screen)，BOOT信息(bootInfo)
- 返回值：空
- 调用示例：无

5.2.3 waitKBCReady()

- 函数声明：`void waitKBCReady();`
- 功能：等待键盘控制电路准备完毕
- 参数：空
- 返回值：空
- 调用示例：

```
/*键盘控制电路初始化函数32位*/
void initKeyboardCMD32(FIFO_BUFFER32 *FIFOBuffer,int dataBase){ //初始化键盘控制电路,确认是否可往键盘控制电路发送信息
    keyFIFOBuffer32 = FIFOBuffer; //通用缓冲区数据存入键盘缓冲区
    keyDataBase = dataBase; //此值或可改常量
    waitKBCReady(); //等待键盘控制电路准备完毕
    io_out8(PORT_KEYCMD,KEYCMD_WRITE_MODE); //将键盘控制电路模式指令0x60写入键盘控制电路端口的寄存器
    waitKBCReady(); //等待键盘控制电路准备完毕
    io_out8(PORT_KEYDAT,KBC_MODE_MOUSE); //将键盘控制电路模式设置为0x47，鼠标模式
}
```

5.2.4 ~~initKeyboardCMD~~()

- 函数声明：`void initKeyboardCMD();`
- 功能：初始化键盘控制电路
- 参数：空
- 返回值：空
- 调用示例：无

5.2.5 ~~enableMouse~~()

- 函数声明：`void enableMouse(MOUSE_CURSOR *mouseCursor);`
- 功能：激活鼠标
- 参数：鼠标指针结构体(mouseCursor)
- 返回值：空
- 调用示例：无

5.2.6 decodeMouse()

- 函数声明：`decodeMouse(MOUSE_CURSOR *mouseCursor,unsigned char data);`
- 功能：激活鼠标
- 参数：鼠标指针结构体(mouseCursor)，鼠标信号(data)
- 返回值：int(解码结果)
- 调用示例：

```
if(decodeMouse(&mouseCursor,data - 512) != 0){
        vmx += mouseCursor.mx;vmy += mouseCursor.my;
        if(vmx < 0){
            vmx = 0;
        }
        if(vmy <0){
            vmy = 0;
        }
        if(vmx > bootInfo->screen_width - 1){
            vmx = bootInfo->screen_width - 1;
        }
```

```
        if(vmy > bootInfo->screen_height - 1){
            vmy = bootInfo->screen_height - 1;
        }
        sprintf(info, "(%3d, %3d)", vmx, vmy);
}
```

5.2.7 enableMouse32()

- 函数声明：`void enableMouse32(FIFO_BUFFER32 *FIFOBuffer,int dataBase,MOUSE_CURSOR *mouseCursor);`
- 功能：初始化键盘、鼠标控制电路,32位
- 参数：，缓冲区(FIFOBuffer)，信号偏移(dataBase)，鼠标指针结构体(mouseCursor)
- 返回值：空
- 调用示例：

```
enableMouse32(&FIFOBuffer32,512,&mouseCursor); //激活鼠标，32位，信号偏移
MOUSE_DATA_BASE = 512
```

5.2.8 initKeyboardCMD32()

- 函数声明：`void enableMouse32(FIFO_BUFFER32 *FIFOBuffer,int dataBase,MOUSE_CURSOR *mouseCursor);`
- 功能：激活键盘、鼠标，32位
- 参数：空
- 返回值：空
- 调用示例：

```
FIFO_BUFFER32 FIFOBuffer32;
initFIFOBuffer32(&FIFOBuffer32,buffer,128,0); //初始化32位通用缓冲区
initKeyboardCMD32(&FIFOBuffer32,256); //初始化键盘控制电路，32位，信号偏移
KEY_DATA_BASE = 256
```

5.2.9 sysHLT()

- 函数声明：`void sysHLT();`
- 功能：激活键盘、鼠标，32位
- 参数：空
- 返回值：空
- 调用示例：无

5.3 task.c 的API

5.3.1 总览

函数名	功能	参数	返回值	备注
initTaskListr	任务初始化	(MEMERY_LIST *)memeryList	TASK	
taskAlloc	任务申请	空	TASK	
taskRun	任务运行	(TASK *)task,(int)level,(int)priority	空	
taskSwitchLimit	任务手动切换	空	空	
taskSwitch	任务自动切换	空	int	
taskSleep	任务休眠	(TASK *)task	空	
addTask	添加任务	(TASK *)task	空	
removeTask	移除任务	(TASK *)task	空	
taskSwitchSub	任务切换子函数	空	空	
idleTask	空闲任务	空	空	
taskNow	获取当前任务	空	TASK	

5.3.2 initTaskList()

- 函数声明：`TASK *initTaskList(MEMERY_LIST *memeryList)`
- 功能：任务初始化
- 参数：内存块表(memeryList)
- 返回值：TASK(任务)
- 调用示例：

```
MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR; //内存表
taskA = initTaskList(memeryList); //初始化任务列表并设置默认任务
```

5.3.3 taskAlloc()

- 函数声明：`TASK *taskAlloc();`
- 功能：任务申请，为任务分配内存
- 参数：空
- 返回值：TASK(任务)
- 调用示例：

```
TASK *task = taskAlloc();
```

5.3.4 taskRun()

- 函数声明：`void taskRun(TASK *task,int level,int priority);`
- 功能：任务启动，指定优先级，指定为0时，不改变优先级；指定优先级队列小于0时不改变队列。
- 参数：任务(task)，优先级队列(level)，优先级(priority)
- 返回值：空
- 调用示例：

```
TASK *task = taskAlloc();
taskRun(task, 2, 2);
```

5.3.5 taskSwitchLimit()

- 函数声明：`void taskSwitchLimit();`
- 功能：任务手动切换。
- 参数：空
- 返回值：空
- 调用示例：无

5.3.6 taskSwitch()

- 函数声明：`void taskSwitchLimit();`
- 功能：任务自动切换。
- 参数：空
- 返回值：空
- 调用示例：无

5.3.7 taskSleep()

- 函数声明：`void taskSleep(TASK *task);`
- 功能：任务休眠。
- 参数：任务(task)
- 返回值：空
- 调用示例：

```
void closeConstask(TASK *task){ //关闭进程
    MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR;
    taskSleep(task); //休眠，使关闭过程中不会切换到此任务，从休眠列表安全删除
    memeryFree4k(memeryList,task->consoleStack, 64 * 1024);
    memeryFree4k(memeryList,(int)task->fifo.buffer, 128 * 4);
    task->flag = 0; //替代taskFree(task);
}
```

5.3.8 addTask()

- 函数声明：`void taskSleep(TASK *task);`
- 功能：添加任务。
- 参数：任务(task)
- 返回值：空
- 调用示例：

```
/*任务启动*/
void taskRun(TASK *task,int level,int priority){
    if(level < 0){ //指定优先级队列小于0，不合法，不改变
        level = task->level;
    }
    if(priority > 0){ //指定的优先级大于0时改变优先级
        task->priority = priority;
    }
    if(task->flag == TASK_FLAG_RUNNING && task->level != level){
        removeTask(task); //从当前优先级队列移除任务，task->flag变为1
    }
    if(task->flag != TASK_FLAG_RUNNING){
        task->level = level; //改变优先级队列
        addTask(task); //将任务添加到新的优先级队列
    }
```

```
    taskList->level_change = 1; //下次任务切换时检查level(??)
}
```

5.3.9 removeTask()

- 函数声明：`void removeTask(TASK *task);`
- 功能：移除任务。
- 参数：任务(task)
- 返回值：空
- 调用示例：

```
/*任务休眠*/
void taskSleep(TASK *task){ //将处于空闲状态的任务从任务列表中移除
    TASK *myTask;
    if(task->flag == TASK_FLAG_RUNNING){
        myTask = taskNow();
        removeTask(task);
        if(task == myTask){ //判断是否自我休眠
            taskSwitchSub(); //自我休眠须先找出需要切换的任务
            myTask = taskNow(); //设定为需要切换的任务
            farjmp(0,myTask->selector); //跳转到该任务
        }
    }
}
```

5.3.10 taskSwitchSub()

- 函数声明：`void taskSwitchSub();`
- 功能：任务休眠。
- 参数：空
- 返回值：空
- 调用示例：

```
/*任务自动切换*/
void taskSwitch(){
    TASK_LEVEL *taskLevel = &taskList->task_level[taskList->level_run];
    TASK *newTask;
    TASK *myTask = taskLevel->tasks[taskLevel->now];
    taskLevel->now++;
    if(taskLevel->now == taskLevel->number){ //防止溢出
        taskLevel->now = 0;
    }
    if(taskList->level_change != 0){
        taskSwitchSub();
        taskLevel = &taskList->task_level[taskList->level_run];
    }
    newTask = taskLevel->tasks[taskLevel->now];
    setTimer(taskTimer,newTask->priority); //设定任务定时器，时间片设置为响应优先级
    if(newTask != myTask){ //两个以上任务才进行切换
        farjmp(0,newTask->selector);
    }
}
```

5.3.11 idleTask()

- 函数声明：`void idleTask();`
- 功能：空闲任务。
- 参数：空
- 返回值：空
- 调用示例：

```
/*任务初始化*/
TASK *initTaskList(MEMERY_LIST *memeryList){
    COUNT i = 0; //计数器
    TASK *task,*idle; //定义任务指针
    SEGMENT_DESCRIPTOR *gdt = (SEGMENT_DESCRIPTOR *)GDT_ADR; //定义GDT
    taskList = (TASK_LIST *)memeryAlloc4k(memeryList,sizeof(TASK_LIST)); //为任务列表申请内存空间
    for(i = 0;i < TASK_MAX;i++){
        taskList->tasks0[i].flag =0; //初始化任务列表状态标志
        taskList->tasks0[i].selector = (TASK_GDT0 + i) * 8; //分配段
        taskList->tasks0[i].tss.ldtr = (TASK_GDT0 + TASK_MAX + i) * 8; //将LDT选择子写入tss.ldtr，创建TSS时可设置LDT
        setSegmDesc(gdt + TASK_GDT0 + i,103,(int)&taskList->tasks0[i].tss,TSS32_PRE); //设定段属性
        setSegmDesc(gdt + TASK_GDT0 + TASK_MAX + i, 15, (int)taskList->tasks0[i].ldt,LDT_PRE); //设定段属性
    }
    for(i = 0;i < LEVEL_MAX;i++){ //遍历优先级列表
        taskList->task_level[i].number = 0;
        taskList->task_level[i].now = 0;
    }
    task = taskAlloc(); //为默认任务申请任务表项
    task->flag = TASK_FLAG_RUNNING; //任务正在运行
    task->priority = 2; //第二优先级，时间片0.02秒
    task->level = 0; //最高优先级队列
    addTask(task); //添加任务到优先级队列
    taskSwitchSub(); //决定要切换的任务
    load_tr(task->selector); //将任务段选择子装载到tr寄存器
    taskTimer = timerAlloc(); //为任务计时器申请空间
    setTimer(taskTimer,task->priority); //设定任务计时器，0.02秒切换一次

    idle = taskAlloc(); //申请任务项
    idle->tss.esp = memeryAlloc4k(memeryList,64 * 1024) + 64 * 1024;
    idle->tss.eip = (int)&idleTask;
    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;
    taskRun(idle,LEVEL_MAX - 1,1); //最低优先级

    return task; //将任务指针返回，该默认任务未本任务管理程序
}
```

5.3.12 taskNow()

- 函数声明：`TASK *taskNow();`
- 功能：获取当前任务。
- 参数：任务(task)
- 返回值：空
- 调用示例：

```
/*任务休眠*/
void taskSleep(TASK *task){ //将处于空闲状态的任务从任务列表中移除
    TASK *myTask;
    if(task->flag == TASK_FLAG_RUNNING){
        myTask = taskNow();
        removeTask(task);
        if(task == myTask){ //判断是否自我休眠
            taskSwitchSub(); //自我休眠须先找出需要切换的任务
            myTask = taskNow(); //设定为需要切换的任务
            farjmp(0,myTask->selector); //跳转到该任务
        }
    }
}
```

5.4 file.c的API

5.4.1 总览

函数名	功能	参数	返回值	备注
readFAT	解码并装载FAT	(int *)fat,(unsigned char *)img	空	
loadFile	装载文件	(int)cluster,(int)size,(char *)file,(int *)fat,(char *)img	空	
searchFile	文件查找（默认）	(char *)fileName	FILE_INFO	
file_search2	文件查找，可指定	(char *)name, (struct FILE_INFO *)fileInfo,(int)max	FILE_INFO	
deleteFile	文件删除	(char *)fileName	int	

5.4.2 readFAT()

- 函数声明：`void readFAT(int *fat,unsigned char *img);`
- 功能：解码并装载FAT。
- 参数：文件分配表(fat)，磁盘地址(img)
- 返回值：空
- 调用示例：无

```
MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR; //内存表
int *fat = (int *)memeryAlloc4k(memeryList,4 * 2880); //为文件分配表申请空间，分配表有两张（一张为备份）
readFAT(fat,(unsigned char *)(DISK_ADR + 0X000200)); //载入文件分配表
```

5.4.3 loadFile()

- 函数声明：`void loadFile(int cluster,int size,char *file,int *fat,char *img);`
- 功能：装载文件。
- 参数：簇(cluster)，大小(size)，文件地址(file)，文件分配表(fat)，磁盘地址(img)
- 返回值：空
- 调用示例：

```
/*文件内容查看*/
void typeCMD(CONSOLE *console,int *fat,char *command){
    int myLen = 5;
    COUNT x = 0;
    COUNT y = 0;
    MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR; //内存表
    FILE_INFO *fileInfo = 0;
    char *file;
    if(strncmp(command,"cat ",4) == 0){
        myLen = 4;
    }
    x = 0;y = 0; //初始化游标
    int flag = 0;
    char fileName[FILE_FULL_NAME_SIZE] = {0};
    char fileExten[FILE_EXTEN_SIZE] = {0};
    for(x = myLen;x < myLen + FILE_NAME_SIZE + 1;x++){
        if(command[x] == '.'){
            strncpy(fileName,command + myLen,x - myLen); //解析文件名
            flag = 1;
            break;
        }
    }
    if(flag == 0){
        strncpy(fileName,command + myLen,FILE_NAME_SIZE + myLen); //解析文件名
    }
    sprintf(fileName, "%-8s",fileName);
    if(flag == 1){
        strncpy(fileExten,command + x + 1,3); //解析拓展名
        sprintf(fileExten, "%-3s",fileExten);
    }
    else{
        sprintf(fileExten,"    \0");
    }
    strcat(fileName,fileExten);
    fileInfo = searchFile(fileName);
    if(fileInfo != 0){
        y = fileInfo->size; //文件长度
        file = (char *)memeryAlloc4k(memeryList,fileInfo->size); //为文件申请空间
        loadFile(fileInfo->cluster,fileInfo->size,file,fat,(char *)(0x003e00 + DISK_ADR)); //装载文件
        console->cursorX = 8; //复位光标
```

```
        for(x = 0;x < y;x++){ //遍历文件内容
            sysprint(console,file[x],1);
        }
        newCMDLine(console);
        memeryFree4k(memeryList,(int)file,fileInfo->size);
    }
    else{
        sysprintl(console,"File not fount.\n");
    }
}
```

5.4.4 searchFile()

- 函数声明：`FILE_INFO *searchFile(char *fileName);`
- 功能：文件查找（默认）。
- 参数：文件名(fileName)
- 返回值：空
- 调用示例：

> 参看loadFile()之调用示例。

5.4.5 file_search2()

- 函数声明：`FILE_INFO *file_search2(char *name, struct FILE_INFO *fileInfo, int max);`
- 功能：文件查找，可指定。
- 参数：文件名(name)，文件信息(fileInfo)，最大查找范围(max)
- 返回值：FILE_INFO(文件信息)
- 调用示例：无

5.4.6 deleteFile()

- 函数声明：`int *deleteFile(char *fileName);`
- 功能：文件删除。
- 参数：文件名(fileName)
- 返回值：int(删除成功返回1，失败返回0)
- 调用示例：

```
/*文件删除*/
void delCMD(CONSOLE *console,int *fat,char *command){
    int myLen = 3;
    if(strncmp(command,"del ",4) == 0){
        myLen = 4;
    }
    else{
        myLen = 3;
    }
    COUNT x = 0;
    COUNT y = 0;
    if(strncmp(command,"cat ",4) == 0){
        myLen = 4;
    }
    x = 0;y = 0; //初始化游标
    int flag = 0;
    char fileName[FILE_FULL_NAME_SIZE] = {0};
    char fileExten[FILE_EXTEN_SIZE] = {0};
```

```
    for(x = myLen;x < myLen + FILE_NAME_SIZE + 1;x++){
        if(command[x] == '.'){
            strncpy(fileName,command + myLen,x - myLen); //解析文件名
            flag = 1;
            break;
        }
    }
    if(flag == 0){
        strncpy(fileName,command + myLen,FILE_NAME_SIZE + myLen); //解析文件名
    }
    sprintf(fileName, "%-8s",fileName);
    if(flag == 1){
        strncpy(fileExten,command + x + 1,3); //解析拓展名
        sprintf(fileExten, "%-3s",fileExten);
    }
    else{
        sprintf(fileExten,"    \0");
    }
    strcat(fileName,fileExten);
    if(deleteFile(fileName) != 0){
        sysprintl(console,"Delete file successful.\n");
    }
    else{
        sysprintl(console,"Delete file fail!\n");
    }
}
```

5.6 graphics.c的API

5.6.1 总览

函数名	功能	参数	返回值	备注
initPalette	画板初始化函数,设置画板前执行CLI屏蔽中断	空	空	
setPalette	设置画板	(int)**start**,(int)**end**,(unsigned char *)**rgb**	空	
pointDraw8	像素点绘制函数，vram为显卡首地址	(char *)**vram**,(short)**scrx**, (PIX_POINT)**point**	空	
boxDraw8	矩形区域绘制函数，vram为显卡首地址	(unsigned char *)**vram**,(int)**scrx**, (int)**bx0**,(int)**by0**,(int)**bx1**, (int)**by1**,(unsigned char)**vcolor**	空	
boxDrawx	批量矩形绘制函数	(char *)**vram**,(short)**scrx**,(BOX *)**box**	空	
initDesk	初始化桌面函数	(char *)**vram**,(short)**scrx**, (short)**scry**	空	
fontDraw8	字体渲染引擎	(char *)**vram**,(int)**scrx**, (int)**fx**, (int)**fy**, (char)**vcolor**, (char *)**font**	空	
fontDraw32	GB2312字体渲染引擎	(char *)**vram**,(int)**scrx**,(int)**fx**, (int)**fy**,(char)**vcolor**,(char *)**font1**,(char *)**font2**	空	
wordsDraw8	字符串批量渲染引擎	(char *)**vram**,(int)**scrx**,(int)**fx**, (int)**fy**,(char)**vcolor**,(unsigned char *)**words**	空	
initMouseCursor8	初始化鼠标指针函数	(char *)**cusorGraph**, (char)**curBackColor**	空	
pictureDraw8	图形渲染函数	(char *)**varm**,(short)**scrx**, (PICTURE)**picture**	空	
windowDraw8	窗口绘制函数	(char *)**vram**,(short)**scrx**, (WINDOW)**window**,(int)**focus**, (int)**area**	空	
createWindow	快速窗口创建	(unsigned char *)**coverBuffer**, (int)**width**,(int)**height**,(char *)**caption**,(int)**type**,(int)**focus**	空	
labelDraw	标签绘制函数	(SHEET *)**cover**,(int)**x**,int**y**, (int)**foreColor**,(int)**backColor**, (char *)**s**,(int)**len**	(空	

函数名	功能	参数	返回值	备注
makeTextBox8	绘制文本框	(SHEET *)**sht**,(int)**x0**,(int)**y0**, (int)**width**,(int)**htight**,(int)**c**	空	
refreshWindowCaption	刷新窗口标题	(unsigned char *)**coverBuffer**, (int)**width**, (int)**height**, (char *)**caption**,(int)**type**,(int)**focus**	空	
syslinewin	窗体直线绘制函数	(SHEET *)**sht**,(int)**x0**,(int)**y0**, (int)**x1**,(int)**y1**,(int)**vcolor**	空	
refreshWindowCaptionx	窗口标题栏刷新函数增强版	(SHEET *)**sht**,(int)**focus**	空	

5.6.2 initPalette()

- 函数声明：`void initPalette();`
- 功能：画板初始化函数,设置画板前执行CLI屏蔽中断。调用`setPalette()`初始化`table_rgb`((系统颜色信息注册表)。
- 参数：空
- 返回值：空
- 调用示例：

```
void NNOSMain(){
    initPalette();  //初始化画板
}
```

> 系统颜色信息注册表`table_rgb`之详细说明，请参看系统常量章节。

5.6.3 setPalette()

- 函数声明：`void setPalette(int start,int end,unsigned char *rgb);`
- 功能：设置画板，解析`table_rgb`(系统颜色信息注册表)，将颜色信息写入到显存。
- 参数：显卡始址(start)，显卡尾址(end)，系统颜色信息注册表地址(rgb)
- 返回值：空
- 调用示例：

```
/*画板初始化*/
void initPalette(){ //画板初始化函数具体实现，table_rgb见sysdef.h
    setPalette(0,COLORNUM - 1,table_rgb); //调用绘图函数
    unsigned char table2[216 * 3];
    int r, g, b;
    for (b = 0; b < 6; b++) {
        for (g = 0; g < 6; g++) {
            for (r = 0; r < 6; r++) {
                table2[(r + g * 6 + b * 36) * 3 + 0] = r * 51;
                table2[(r + g * 6 + b * 36) * 3 + 1] = g * 51;
                table2[(r + g * 6 + b * 36) * 3 + 2] = b * 51;
            }
        }
```

```
    }
    setPalette(COLORNUM,242, table2);
}
```

5.6.4 pointDraw8()

- 函数声明：`void pointDraw8(char *vram,short scrx,PIX_POINT point);`
- 功能：在画板上绘制一个像素大小的点。
- 参数：显存地址(vram)，屏幕宽度(像素)(scrx)，点实体(point)
- 返回值：空
- 调用示例：无

> 本系统绘图函数尚待统一，为保证底层函数兼容性，之后的版本中有可能取消参数中的点实体，改为两端点坐标。
>
> 函数`pointDraw8`有可能在函数名不变的情况下，修改参数。
>
> 在本注释删除前，不建议使用该函数。
>
> 附：
>
> ```
> /*像素点结构体*/
> typedef struct PIX_POINT{
>  int px;                 //点横坐标
>  int py;                 //点纵坐标
>  unsigned char vcolor;   //填充色
> }PIX_POINT;
> ```

5.6.5 boxDraw8()

- 函数声明：`void boxDraw8(unsigned char *vram,int scrx,int bx0,int by0,int bx1,int by1,unsigned char vcolor);`
- 功能：在画板上绘制一个矩形。
- 参数：显存地址(vram)，屏幕宽度(像素)(scrx)，左上角横坐标(bx0)，左上角纵坐标(by0)，右下角横坐标(bx1)，右下角纵坐标(by1)，矩形颜色(vcolor)
- 返回值：空
- 调用示例：

```
void labelDraw(SHEET *cover,int x,int y,int foreColor,int backColor,char *s,int len){
    TASK *task = taskNow();
    boxDraw8(cover->buf, cover->bxsize, x, y, x + len * 8 - 1, y + 15, backColor);
    if (task->lang_mode != 0 && task->lang_byte != 0) {
        wordsDraw8(cover->buf,cover->bxsize,x,y,foreColor,s);
        sheet_refresh(cover, x - 8, y, x + len * 8, y + 16);
    } else {
        wordsDraw8(cover->buf,cover->bxsize,x,y,foreColor,s);
        sheet_refresh(cover,x,y,x + len * 8,y + 16);
    }
}
```

> 横纵坐标以画板左上角为原点。

5.6.6 boxDrawx()

- 函数声明：`void boxDrawx(char *vram,short scrx,BOX *box);`
- 功能：批量绘制矩形,该函数会将box数组中的每个元素遍历并绘制在画板上。可用于简单窗口的拼接构建。
- 参数：显存地址(vram)，屏幕宽度(像素)(scrx)，色块数组首地址(box)
- 返回值：空
- 调用示例：

```
/*桌面初始化*/
void initDesk(char *vram,short scrx,short scry){
    BOX deskWindows[] = {
        {0,0,scrx - 1,scry - 15,COL8_004276,"DeskFore"}, //桌面背景色
        {0,scry - 15,scrx - 1,scry - 1,COL8_005B9E,"TaskBar"}, //任务栏
        {0,scry - 1,15,scry - 15,COL8_E1E1E1,"StartDrw"}, //开始菜单渲染
        {0,scry - 15,15,scry - 2,COL8_0078D7,"StartBtn"}, //开始菜单
        {scrx-4,scry-15,scrx-4,scry,COL8_E1E1E1,"DeLine"}, //任务栏分割线
        {5,5,25,25,COL8_E1E1E1,"IconDrw"}, //桌面图标渲染
        {6,6,24,24,COL8_0078D7,"IconInit"}, //桌面图标
        {-1,0,0,0,0,"EndFlag"} //结束标记
    };
    boxDrawx(vram,scrx,deskWindows);
}
```

> 每个传入的`box`数组都应该包含一个结束标记，该结束标记为`box`数组的最后一个元素，结束标记的色块标识名`box_name`值必须为`EndFlag`，其余参数可任意设置。结束标记不会被绘制，仅用作批量绘制色块时的识别标识。
>
> 附：
>
> ```
> /*色块结构体*/
> typedef struct BOX{
>  int bx0;                //矩形横向起点
>  int by0;                //矩形纵向起点
>  int bx1;                //矩形横向终点
>  int by1;                //矩形纵向终点
>  unsigned char vcolor;   //填充色
>  char box_name[8]; //色块标识名
> }BOX;
> ```

5.6.7 initDesk()

- 函数声明：`void initDesk(char *vram,short scrx,short scry);`
- 功能：初始化桌面。待完善。
- 参数：显存地址(vram)，屏幕宽度(像素)(scrx)，屏幕高度(像素)(scry)
- 返回值：空
- 调用示例：

```
BOOT_INFO *bootInfo = (BOOT_INFO *) BOOTINFO_ADR; //设置BOOT信息存储位置
unsigned char *buf_back, buf_mouse[256]; //桌面背景、鼠标指针、窗口信息数组
initDesk(buf_back,bootInfo->screen_width, bootInfo->screen_height); //初始化桌面
```

> 该函数目前仅完成了桌面底色和任务栏的覆盖，有待完善。之后的开发中计划移植图像查看器至桌面初始化API中，使之支持以图片为桌面背景。

5.6.8 fontDraw8()

- 函数声明：`void fontDraw8(char *vram,int scrx, int fx, int fy, char vcolor, char *font);`
- 功能：字体渲染引擎。
- 参数：显存地址(vram)，屏幕宽度(像素)(scrx)，横坐标(fx)，纵坐标(fy)，字体颜色(vcolor)，字符地址(font)
- 返回值：空
- 调用示例：

```
void wordsDraw8(char *vram,int scrx,int fx,int fy,char vcolor,unsigned char *words){ //批量写字符
    extern char fonbase[4096]; //引入字库
    struct TASK *task = taskNow(); //获取当前任务
    char *fonts = (char *) *((int *) 0x0fe8), *font; //日文字库地址
    int k, t; //k，存放区号；t，存放点号。存放减1后的值（方便数组运用）

    if (task->lang_mode == 0) {
        for (; *words != 0x00; words++) {
            fontDraw8(vram, scrx, fx, fy, vcolor, fonbase + *words * 16);
            fx += 8;
        }
    }
    if (task->lang_mode == 1) {
        for (; *words != 0x00; words++) {
            if (task->lang_byte == 0) {
                if ((0x81 <= *words && *words <= 0x9f) || (0xe0 <= *words && *words <= 0xfc)) {
                    task->lang_byte = *words;
                }
                else {
                    fontDraw8(vram, scrx, fx, fy, vcolor, fonts + *words * 16);
                }
            }
            else {
                if (0x81 <= task->lang_byte && task->lang_byte <= 0x9f) {
                    k = (task->lang_byte - 0x81) * 2;
                } else {
                    k = (task->lang_byte - 0xe0) * 2 + 62;
                }
                if (0x40 <= *words && *words <= 0x7e) {
                    t = *words - 0x40;
                } else if (0x80 <= *words && *words <= 0x9e) {
                    t = *words - 0x80 + 63;
                } else {
                    t = *words - 0x9f;
                    k++;
                }
                task->lang_byte = 0;
                font = fonts + 256 * 16 + (k * 94 + t) * 32;
                fontDraw8(vram, scrx, fx - 8, fy, vcolor, font     );   //左半部分
```

```
                fontDraw8(vram, scrx, fx    , fy, vcolor, font + 16);   //右半部分
            }
            fx += 8;
        }
    }
    if (task->lang_mode == 2) {
        for (; *words != 0x00; words++) {
            if (task->lang_byte == 0) {
                if (0x81 <= *words && *words <= 0xfe) {
                    task->lang_byte = *words;
                } else {
                    fontDraw8(vram, scrx, fx, fy, vcolor, fonts + *words * 16);
                }
            } else {
                k = task->lang_byte - 0xa1;
                t = *words - 0xa1;
                task->lang_byte = 0;
                font = fonts + 256 * 16 + (k * 94 + t) * 32;
                fontDraw8(vram, scrx, fx - 8, fy, vcolor, font     );   //左半部分
                fontDraw8(vram, scrx, fx    , fy, vcolor, font + 16);   //右半部分
            }
            fx += 8;
        }
    }
    if (task->lang_mode == 3) {
        for (; *words != 0x00; words++) {
            if (task->lang_byte == 0) {
                if (0xa1 <= *words && *words <= 0xfe) {
                    task->lang_byte = *words;
                } else {
                    fontDraw8(vram, scrx, fx, fy, vcolor, fonbase + *words * 16);

                }
            }
            else{
                k = task->lang_byte - 0xa1;
                t = *words - 0xa1;
                task->lang_byte = 0;
                font = fonts + (k * 94 + t) * 32;
                fontDraw32(vram,scrx,fx-8,fy,vcolor,font,font+16);
            }
            fx += 8;
        }
    }
}
```

> 横坐标、纵坐标，以屏幕左上角为原点，定位于字符左上角。

5.6.9 fontDraw32()

- 函数声明：`void fontDraw32(char *vram,int scrx,int fx,int fy,char vcolor,char *font1,char *font2);`
- 功能：字体渲染引擎。
- 参数：显存地址(vram)，屏幕宽度(像素)(scrx)，横坐标(fx)，纵坐标(fy)，字体颜色(vcolor)，字符串地址(font)，字符地址2(font1)，字符地址(font2)
- 返回值：空
- 调用示例：

```
void wordsDraw8(char *vram,int scrx,int fx,int fy,char vcolor,unsigned char *words){ //批量写字符
    extern char fonbase[4096]; //引入字库
    struct TASK *task = taskNow(); //获取当前任务
    char *fonts = (char *) *((int *) 0x0fe8), *font; //日文字库地址
    int k, t; //k，存放区号；t，存放点号。存放减1后的值（方便数组运用）

    if (task->lang_mode == 0) {
        for (; *words != 0x00; words++) {
            fontDraw8(vram, scrx, fx, fy, vcolor, fonbase + *words * 16);
            fx += 8;
        }
    }
    if (task->lang_mode == 1) {
        for (; *words != 0x00; words++) {
            if (task->lang_byte == 0) {
                if ((0x81 <= *words && *words <= 0x9f) || (0xe0 <= *words && *words <= 0xfc)) {
                    task->lang_byte = *words;
                }
                else {
                    fontDraw8(vram, scrx, fx, fy, vcolor, fonts + *words * 16);
                }
            }
            else {
                if (0x81 <= task->lang_byte && task->lang_byte <= 0x9f) {
                    k = (task->lang_byte - 0x81) * 2;
                } else {
                    k = (task->lang_byte - 0xe0) * 2 + 62;
                }
                if (0x40 <= *words && *words <= 0x7e) {
                    t = *words - 0x40;
                } else if (0x80 <= *words && *words <= 0x9e) {
                    t = *words - 0x80 + 63;
                } else {
                    t = *words - 0x9f;
                    k++;
                }
                task->lang_byte = 0;
                font = fonts + 256 * 16 + (k * 94 + t) * 32;
                fontDraw8(vram, scrx, fx - 8, fy, vcolor, font     );   //左半部分
                fontDraw8(vram, scrx, fx    , fy, vcolor, font + 16);   //右半部分
            }
```

```
            fx += 8;
        }
    }
    if (task->lang_mode == 2) {
        for (; *words != 0x00; words++) {
            if (task->lang_byte == 0) {
                if (0x81 <= *words && *words <= 0xfe) {
                    task->lang_byte = *words;
                } else {
                    fontDraw8(vram, scrx, fx, fy, vcolor, fonts + *words *
16);
                }
            } else {
                k = task->lang_byte - 0xa1;
                t = *words - 0xa1;
                task->lang_byte = 0;
                font = fonts + 256 * 16 + (k * 94 + t) * 32;
                fontDraw8(vram, scrx, fx - 8, fy, vcolor, font     );   //左
半部分
                fontDraw8(vram, scrx, fx    , fy, vcolor, font + 16);   //右
半部分
            }
            fx += 8;
        }
    }
    if (task->lang_mode == 3) {
        for (; *words != 0x00; words++) {
            if (task->lang_byte == 0) {
                if (0xa1 <= *words && *words <= 0xfe) {
                    task->lang_byte = *words;
                } else {
                    fontDraw8(vram, scrx, fx, fy, vcolor, fonbase + *words *
16);

                }
            }
            else{
                k = task->lang_byte - 0xa1;
                t = *words - 0xa1;
                task->lang_byte = 0;
                font = fonts + (k * 94 + t) * 32;
                fontDraw32(vram,scrx,fx-8,fy,vcolor,font,font+16);
            }
            fx += 8;
        }
    }
}
```

5.6.10 wordsDraw8**()

- 函数声明：`void wordsDraw8(char *vram,int scrx,int fx,int fy,char vcolor,unsigned char *words);`
- 功能：字符串批量渲染引擎。
- 参数：显存地址(vram)，屏幕宽度(像素)(scrx)，横坐标(fx)，纵坐标(fy)，字体颜色(vcolor)，字符串地址(words)
- 返回值：空
- 调用示例：

```
BOOT_INFO *bootInfo = (BOOT_INFO *) BOOTINFO_ADR; //设置BOOT信息存储位置
unsigned char *buf_back; //桌面背景数组
int centerX = (bootInfo->screen_width - 16) / 2; //屏幕中心点横坐标
int centerY = (bootInfo->screen_height - 28 - 16) / 2; //屏幕中心点纵坐标
wordsDraw8(buf_back,bootInfo->screen_width,centerX - 30 - 1,centerY -
1,COL8_000000,NNOS_VERSION); //字体渲染
```

5.6.11 initMouseCursor8()

- 函数声明：`void initMouseCursor8(char *cusorGraph,char curBackColor);`
- 功能：初始化鼠标指针。
- 参数：鼠标指针点阵(cusorGraph)，指针背景色(像素)(curBackColor)
- 返回值：空
- 调用示例：

```
BOOT_INFO *bootInfo = (BOOT_INFO *) BOOTINFO_ADR; //设置BOOT信息存储位置
int centerX = (bootInfo->screen_width - 16) / 2; //屏幕中心点横坐标
int centerY = (bootInfo->screen_height - 28 - 16) / 2; //屏幕中心点纵坐标
MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR; //内存表
initMemeryManager(memeryList);  //初始化内存管理系统
memeryFree(memeryList,0x00001000,0x0009e000); //设置空闲内存,0x00001000 -
0x0009efff
memeryFree(memeryList,0x00400000,memeryList->total_size - 0x00400000); //设
置空闲内存
SHTCTL *shtctl; //图层列表
shtctl = shtctl_init(memeryList,bootInfo->vram_base,bootInfo-
>screen_width,bootInfo->screen_height); //为图层表申请内存空间
unsigned char buf_mouse[256]; //鼠标指针数组
sht_mouse = sheet_alloc(shtctl); //鼠标图层分配
sheet_setbuf(sht_mouse, buf_mouse, 16,16,99); //鼠标图层设定
initMouseCursor8(buf_mouse,99); //初始化鼠标指针图像点阵
```

5.6.12 pictureDraw8**()

- 函数声明：`void pictureDraw8(char *varm,short scrx,PICTURE picture);`
- 功能：图形渲染
- 参数：显存地址(vram)，屏幕宽度(像素)(scrx)，图形结构体（picture)
- 返回值：空
- 调用示例：无

5.6.13 windowDraw8()

- 函数声明：`void windowDraw8(char *vram,short scrx,WINDOW window,int focus,int area);`
- 功能：窗口绘制
- 参数：显存地址(vram)，屏幕宽度(像素)(scrx)，窗口结构体(window)，是否获得焦点(focus)，刷新区域(area)
- 返回值：空
- 调用示例：

> 聚焦、失焦两种状态，窗口标题栏颜色不同。参数`focus`设置所绘窗口是否获得焦点，1为获得焦点，2为失去焦点。
>
> 窗口聚焦、失焦状态可能频繁切换，参数`area`用于对窗口标题栏刷新。

5.6.14 createWindow()

- 函数声明：`void createWindow(unsigned char *coverBuffer, int width, int height, char *caption,int type,int focus);`
- 功能：快速窗口创建
- 参数：图层缓冲区(coverBuffer)，窗口宽度(width)，窗口高度(height)，窗口标题(caption)，窗口结构体(window)，窗口类型(type)，是否获得焦点(focus)
- 返回值：空
- 调用示例：

```
MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR;
nsigned char *buf = (unsigned char *)memeryAlloc4k(memeryList,CONSOLE_WIDTH * CONSOLE_HEIGHT);
createWindow(buf,CONSOLE_WIDTH,CONSOLE_HEIGHT,"Console",CONTROL_WINDOW,0); //创建控制台窗口
```

> 目前窗口类型共4中，可在`sysdef.h`中查看。
>
> 附：窗口类型
>
> ```
> #define CONTROL_WINDOW 1 //控制台窗口类型号
> #define NOMAL_WINDOW 2 //通用窗口类型号
> #define INFO_WINDOW 3 //信息提示窗口类型号
> #define TEXT_WINDOW 4 //文本输入窗口类型号
> ```

5.6.15 labelDraw()

- 函数声明：`void labelDraw(SHEET *cover,int x,int y,int foreColor,int backColor,char *s,int len);`
- 功能：标签绘制
- 参数：图层(cover)，横坐标(x)，纵坐标(y)，前景色(foreColor)，背景色(backColor)，字符串(s)，字符串长度(len)
- 返回值：空
- 调用示例：

```
SHEET *sht_back;//图层
sheet_setbuf(sht_back, buf_back, bootInfo->screen_width, bootInfo->screen_height, -1);  //设置桌面背景图层
labelDraw(sht_back,0,64,COL8_FFFFFE,DESK_BCOLOR,"10(sec)",10); //绘制10秒提示
```

5.6.16 makeTextBox8()

- 函数声明：`void makeTextBox8(SHEET *sht,int x0,int y0,int width,int height,int c);`
- 功能：绘制文本框
- 参数：图层(cover)，横坐标(x0)，纵坐标(y0)，宽度(width)，高度(height)，字符串(s)，边框颜色(c)
- 返回值：空
- 调用示例：无

5.6.17 refreshWindowCaption()

- 函数声明：`void refreshWindowCaption(unsigned char *coverBuffer, int width, int height, char *caption,int type,int focus);`
- 功能：刷新窗口标题
- 参数：图层缓冲区(coverBuffer)，宽度(width)，高度(height)，标题(caption)，类型(type)，是否获得焦点(focus)
- 返回值：空
- 调用示例：无

5.6.18 syslinewin()

- 函数声明：`void syslinewin(SHEET *sht,int x0,int y0,int x1,int y1,int vcolor);`
- 功能：窗体直线绘制
- 参数：图层(sht)，A端横坐标(x0)，A端纵坐标(y0)，B端横坐标(x1)，1端纵坐标(y1)，颜色(c)
- 返回值：空
- 调用示例：无

> 线宽默认1像素。

5.6.19 refreshWindowCaptionx()

- 函数声明：`void refreshWindowCaptionx(SHEET *sht,int focus);`
- 功能：窗口标题栏刷新函数增强版
- 参数：图层(sht)，是否获得焦点(focus)
- 返回值：空
- 调用示例：

```
void keywinOff(SHEET *keyWin){
    refreshWindowCaptionx(keyWin,1);
    if ((keyWin->flags & 0x20) != 0) {
        putFIFOBuffer32(&keyWin->task->fifo, 3);
    }
}

void keywinOn(SHEET *keyWin){
    refreshWindowCaptionx(keyWin,0);
    if ((keyWin->flags & 0x20) != 0) {
        putFIFOBuffer32(&keyWin->task->fifo, 2);
    }
}
```

5.7 bootpack.c的API

> `bootpack.c`为系统主程序。

5.7.1 总览

函数名	功能	参数	返回值	备注
keywinOff	键盘锁定，开始从缓冲区取数据（存疑）	(SHEET *)keyWin	空	
keywinOn	键盘开启，开始向缓冲区写入数据（存疑）	(SHEET *)keyWin	空	
~~change_wtitle8~~	切换窗口	(SHEET *)sht，(char)act	空	已废弃，使用切换图层替代
openConsole	打开控制台窗口	(SHEET *)shtctl	SHEET	
closeConstask	结束控制台任务	(TASK *)task	空	
closeConsole	关闭控制台窗口	(SHEET)sht	空	
newConsole	创建新的控制台窗口	(SHEET *)shtctl,(SHEET *)keyWin,(int)scrx,(int)scry	空	

5.7.2 keywinOff()

- 函数声明：`void keywinOff(SHEET *keyWin);`
- 功能：键盘锁定，开始从缓冲区取数据（存疑）。
- 参数：输入框首地址(keyWin)
- 返回值：空
- 调用示例：

```
if(data == 0x0f + 256){ //TAB
    keywinOff(keyWin);
    j = keyWin->height - 1;
    if (j == 0) {
        j = shtctl->top - 1;
    }
    keyWin = shtctl->sheets[j];
    keywinOn(keyWin);
}
```

5.7.3 keywinOn()

- 函数声明：`void keywinOn(SHEET *keyWin);`
- 功能：键盘开启，开始向缓冲区写入数据（存疑）。
- 参数：输入框首地址(keyWin)
- 返回值：空
- 调用示例：

> 参看14.1.2节。理论上每个keywinOff后面都应该有至少一个keywinOn。
>
> 该函数功能暂不明确，若您能补充相关信息，请在QQ群内讨论或者发信息至邮箱tianhehechu@qq.com。

5.7.4 ~~change_wtitle8()~~

- 函数声明：`void change_wtitle8(SHEET *sht, char act);`
- 功能：切换窗口。
- 参数：图层(sht)，动作(act)
- 返回值：空
- 调用示例：无

> 本函数已废弃并移除，不可使用。

5.7.5 openConsole()

- 函数声明：`SHEET *openConsole(SHTCTL *shtctl);`
- 功能：打开控制台窗口。
- 参数：图层列表(shtctl)
- 返回值：空
- 调用示例：

```
void newConsole(SHTCTL *shtctl,SHEET *keyWin,int scrx,int scry){
    if(keyWin != 0){
        keywinOff(keyWin);
    }
    keyWin = openConsole(shtctl);
    sheet_slide(keyWin,CX,CY);
    sheet_updown(keyWin,shtctl->top);
    CX += 8;CY += 28;
    if(CX + CONSOLE_WIDTH >= scrx){
        CX = 8;
    }
    if(CY + CONSOLE_HEIGHT >= scry){
        CY = 8;
    }
    keywinOn(keyWin);
}
```

5.7.6 closeConstask()

- 函数声明：`void closeConstask(TASK *task);`
- 功能：结束控制台任务。
- 参数：任务列表(task)
- 返回值：空
- 调用示例：

```
void closeConsole(SHEET *sht){ //关闭窗口图层
    MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR;
    TASK *task = sht->task;
    memeryFree4k(memeryList,(int)sht->buf, 256 * 165);
    sheet_free(sht);
    closeConstask(task);
}
```

5.7.7 closeConsole()

- 函数声明：`void closeConsole(SHEET *sht);`
- 功能：关闭控制台窗口。
- 参数：图层(sht)
- 返回值：空
- 调用示例：

```
//完整代码见bootpack.c中的NNOSMain()函数
//省略前驱代码
if(data >= 768 && data <=1023){ //关闭命令行
    closeConsole(shtctl->sheets0 + (data - 768)); //根据句柄关闭图层，偏移为768
}
//省略后继代码
```

5.7.8 newConsole()

- 函数声明：`void newConsole(SHTCTL *shtctl,SHEET *keyWin,int scrx,int scry);`
- 功能：创建新的控制台窗口。
- 参数：图层列表(shtctl)，图层(keyWin)，屏幕宽度(scrx)，屏幕高度(scry)
- 返回值：空
- 调用示例：无

六、扩展层API详解

> 本层API的声明，全部在`nnos.h`中。
>
> `nnos.h`为系统核心库的头文件，包含全部引导层和基础层API。对于osfun的部分，本节不再重复解析；
>
> `nnos.h`引用两个核心库，`sysdef.h`（系统常量表）和`systructual.h`（系统结构体库）。
>
> 【注意】本层全部API皆为系统级，仅可出现在应用层（不含应用层）以下，否则将出现安全问题和不可预知的错误。

6.1 cover.c的API

> 【注意】`cover.c`中所有API，皆处于调试阶段，不可使用。
>
> 在本模块调试稳定前，请使用`others.c`中的相应API替代本模块API。

6.1.1 总览

函数名	功能	参数	返回值	备注
initCoverList	图层初始化	(MEMERY_LIST *)memeryList,(SCREEN)screen	空	shtctl_init
coverAlloc	图层内存分配	(MEMERY_LIST *)memeryList,(COVER_LIST *)coverList	空	sheet_alloc
setCover	图层设定	(COVER *)cover,(PICTURE)picture,(int)transColor	空	sheet_setbuf
updateCover	图层更新	(COVER_LIST *)coverList,(COVER *)cover,(int)order	空	sheet_updown
coverRefresh	图层刷新	(COVER_LIST *)coverList	空	sheet_refresh
coverMove	图层移动	(COVER_LIST *)coverList,(COVER *)cover,(int)vx,(int)vy	空	sheet_slide
coverFree	图层释放	(MEMERY_LIST *)memeryList,(COVER_LIST *)coverList,(COVER *)cover	空	sheet_free

6.1.2 initCoverList()

- 函数声明：`COVER_LIST *initCoverList(MEMERY_LIST *memeryList,SCREEN screen);`
- 功能：图层初始化
- 参数：内存块表(memeryList)，屏幕信息(screen)
- 返回值：COVER_LIST(图层列表)
- 调用示例：无

> 暂以`shtctl_init()`代替

6.1.3 coverAlloc()

- 函数声明：`COVER *coverAlloc(MEMERY_LIST *memeryList,COVER_LIST *coverList);`
- 功能：图层内存分配
- 参数：内存块表(memeryList)，图层列表(coverList)
- 返回值：COVER(图层)
- 调用示例：无

> 暂以`sheet_alloc()`代替

6.1.4 setCover()

- 函数声明：`void setCover(COVER *cover,PICTURE picture,int transColor);`
- 功能：图层设定
- 参数：图层(cover)，图像(picture),是否透明(transColor)
- 返回值：空
- 调用示例：无

> 暂以`sheet_setbuf()`代替

6.1.5 updateCover()

- 函数声明：`void updateCover(COVER_LIST *coverList,COVER *cover,int order);`
- 功能：图层更新
- 参数：图层列表(coverList)，图层(cover)，图层层级(order)
- 返回值：空
- 调用示例：无

> 暂以`sheet_updown`代替

6.1.6 coverRefresh()

- 函数声明：`void coverRefresh(COVER_LIST *coverList);`
- 功能：图层刷新
- 参数：图层列表(coverList)
- 返回值：空
- 调用示例：无

> 暂以`sheet_refresh()`代替

6.1.7 coverMove()

- 函数声明：`void coverMove(COVER_LIST *coverList,COVER *cover,int vx,int vy);`
- 功能：图层移动
- 参数：图层列表(coverList)，图层(cover)，横坐标变量(vx)，纵坐标变量(vy)
- 返回值：空
- 调用示例：无

> 暂以`sheet_slide()`代替

6.1.8 coverFree()

- 函数声明：`void coverFree(MEMERY_LIST *memeryList,COVER_LIST *coverList,COVER *cover);`
- 功能：图层释放
- 参数：内存块表(memeryList)，图层列表(coverList)，图层(cover)
- 返回值：空
- 调用示例：无

> 暂以`sheet_free()`代替

6.2 window.c 的API

6.2.1 总览

函数名	功能	参数	返回值	备注
initWindowList	初始化窗口列表	(WINDOW_LIST *)windowList	空	
windowAlloc	窗口申请	空	WINDOW	
setWindowBox	窗口设定	(WINDOW_LIST *)windowList,(WINDOW *)window,(BOX *)box	空	
setWindowCaption	窗口标题设定	(WINDOW *)window,(unsigned char)backColor	空	
setWindowFocus	窗口焦点设定	(WINDOW_LIST *)windowList,(WINDOW *)window	空	
getWindow	获取窗口句柄	(WINDOW_LIST *)windowList,(int)id	WINDOW	

6.2.2 initWindowList()

- 函数声明：`void initWindowList(WINDOW_LIST *windowList);`
- 功能：初始化窗口列表。
- 参数：窗口列表(windowList)
- 返回值：空
- 调用示例：无

6.2.3 windowAlloc()

- 函数声明：`WINDOW *windowAlloc();`
- 功能：窗口申请，为窗口分配内存。
- 参数：空
- 返回值：WINDOW
- 调用示例：无

6.2.4 setWindowBox()

- 函数声明：`void setWindowBox(WINDOW_LIST *windowList,WINDOW *window,BOX *box);`
- 功能：窗口设定。
- 参数：窗口列表(windowList)，窗口(window)，色块(box)
- 返回值：空
- 调用示例：无

6.2.5 setWindowCaptiont()

- 函数声明：`void setWindowCaption(WINDOW *window,unsigned char backColor);`
- 功能：窗口标题设定。
- 参数：窗口(window)，背景色(backColor)
- 返回值：空
- 调用示例：无

6.2.6 setWindowFocus()

- 函数声明：`void setWindowFocus(WINDOW_LIST *windowList,WINDOW *window);`
- 功能：窗口焦点设定。
- 参数：窗口列表(windowList)，窗口(window)
- 返回值：空
- 调用示例：无

6.2.7 getWindowt()

- 函数声明：`WINDOW *getWindow(WINDOW_LIST *windowList,int id);`
- 功能：获取窗口句柄。
- 参数：窗口列表(windowList)，窗口句柄(id)
- 返回值：空
- 调用示例：无

6.3 others.c的API

6.3.1 总览

函数名	功能	参数	返回值	备注
shtctl_init	图层初始化	(MEMERY_LIST *)memman,(unsigned char *)vram,(int)xsize,(int)ysize	SHTCTL	
sheet_alloc	图层申请	(SHTCTL *)ctl	SHEET	
sheet_setbuf	图层设定	(SHEET *)sht,(unsigned char *)buf,(int)xsize,(int)ysize,(int)col_inv	空	
sheet_updown	图层更新	(SHEET *)sht,(int)height	空	
sheet_refresh	图层刷新	(SHEET *)sht,(int)bx0,(int)by0,(int)bx1,(int)by1	空	
sheet_refreshsub	图层刷新子函数	(SHEET *)ctl,(int)vx0,(int)vy0,(int)vx1,(int)vy1,(int)h0,(int)h1	空	
sheet_refreshmap	图层热刷新	(SHEET *)ctl,(int)vx0,(int)vy0,(int)vx1,(int)vy1,(int)h0	空	
sheet_slide	图层移动	(SHEET *)ctl,(int)vx0,(int)vy0	空	
sheet_free	图层释放	(SHEET *)ctl	空	

6.3.2 shtctl_init()

- 函数声明：`SHTCTL *shtctl_init(MEMERY_LIST *memman, unsigned char *vram, int xsize, int ysize);`
- 功能：图层初始化。
- 参数：内存块表(memman)，显存地址(vram)，图层宽度(xsize)，图层高度(ysize)
- 返回值：SHTCTL(图层表)
- 调用示例：

```
MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR; //内存表
shtctl = shtctl_init(memeryList,bootInfo->vram_base,bootInfo->screen_width,bootInfo->screen_height); //为图层表申请内存空间
```

6.3.3 sheet_alloc()

- 函数声明：`SHEET *sheet_alloc(SHTCTL *ctl);`
- 功能：图层申请，为图层分配内存。
- 参数：图层列表(ctl)
- 返回值：SHEET(图层)
- 调用示例：

```
SHTCTL *shtctl; //图层列表
shtctl = shtctl_init(memeryList,bootInfo->vram_base,bootInfo->screen_width,bootInfo->screen_height); //为图层表申请内存空间
sht_back  = sheet_alloc(shtctl); //分配背景图层
buf_back  = (unsigned char *) memeryAlloc4k(memeryList, bootInfo->screen_width * bootInfo->screen_height); //为桌面背景图层缓冲区申请空间
sheet_setbuf(sht_back, buf_back, bootInfo->screen_width, bootInfo->screen_height, -1);  //设置桌面背景图层
```

6.3.4 sheet_setbuf()

- 函数声明：`void sheet_setbuf(SHEET *sht, unsigned char *buf, int xsize, int ysize, int col_inv);`
- 功能：图层设定。
- 参数：图层(sht)，缓冲区(buf)，图层宽度(xsize)，图层高度(ysize)，图层颜色(col_inv)
- 返回值：空
- 调用示例：

> 参看sheet_alloc()之调用示例。
>
> 图层颜色参数`col_inv`为`-1`时，图层背景透明。

6.3.5 sheet_updown()

- 函数声明：`void sheet_updown(SHEET *sht, int height);`
- 功能：更新图层，调整图层层级。
- 参数：图层(sht)，层级(height)
- 返回值：空
- 调用示例：

```
/*图层调整*/
sheet_slide(sht_back,0,0); //桌面背景图层复位
sheet_slide(sht_mouse,vmx,vmy); //鼠标图层复位
sheet_updown(sht_back,0); //图层顺序设定
sheet_updown(sht_mouse,2);
```

6.3.6 sheet_refresh()

- 函数声明：`void sheet_refresh(SHEET *sht, int bx0, int by0, int bx1, int by1);`
- 功能：图层刷新。
- 参数：图层(sht)，左上角横坐标(bx0)，左上角纵坐标(by0)，左下角横坐标(bx1)，左下角纵坐标(by1)
- 返回值：空
- 调用示例：

```
void refreshWindowCaptionx(SHEET *cover,int focus){
    int x,y,xsize;
    char color,oldColor,newColor,boldcolor,bnewColor;
    xsize = cover->bxsize;
    if(focus == 0){
        newColor = COL8_0078D7;
        oldColor  = COL8_E1E1E1;
        bnewColor = COL8_E81123;
        boldcolor = COL8_E1E1E1;
    }
    else{
        newColor = COL8_E1E1E1;
        oldColor = COL8_0078D7;
        bnewColor = COL8_E1E1E1;
        boldcolor = COL8_E81123;
    }
    for (y = 1;y <= 21;y++) {
        for (x = 1; x <= xsize - 4;x++){
            color = cover->buf[y * xsize + x];
            if (color == oldColor && x <= xsize - 22){
                color = newColor;
            }
            else if(color == boldcolor) {
                color = bnewColor;
            }
            cover->buf[y * xsize + x] = color;
        }
    }
    sheet_refresh(cover,1,1, xsize,22);
}
```

6.3.7 heet_refreshsub()

- 函数声明：`void sheet_refreshsub(SHTCTL *ctl, int vx0, int vy0, int vx1, int vy1, int h0, int h1);`
- 功能：图层刷新子函数。
- 参数：图层(sht)，左上角横坐标变量(vx0)，左上角纵坐标变量(vy0)，左下角横坐标变量(vx1)，左下角纵坐标变量(vy1)，原高度(h0)，新高度(h1)
- 返回值：空
- 调用示例：

```
/*图层刷新*/
void sheet_refresh(SHEET *sht,int bx0,int by0,int bx1,int by1){
    if(sht->height >= 0){
        sheet_refreshsub(sht->ctl,sht->vx0 + bx0,sht->vy0 + by0,sht->vx0 + bx1,sht->vy0 + by1,sht->height,sht->height);
    }
    return;
}
```

6.3.8 sheet_refreshmap()

- 函数声明：`void sheet_refresh(SHEET *sht, int bx0, int by0, int bx1, int by1);`
- 功能：图层热刷新，即只刷新发生变化的像素。
- 参数：图层(sht)，左上角横坐标变量(vx0)，左上角纵坐标变量(vy0)，左下角横坐标变量(vx1)，左下角纵坐标变量(vy1)，原高度(h0)
- 返回值：空
- 调用示例：

```
/*图层移动*/
void sheet_slide(SHEET *sht,int vx0,int vy0){
    SHTCTL *ctl = sht->ctl;
    int old_vx0 = sht->vx0,old_vy0 = sht->vy0;
    sht->vx0 = vx0;
    sht->vy0 = vy0;
    if (sht->height >= 0) {
        sheet_refreshmap(ctl,old_vx0,old_vy0,old_vx0 + sht->bxsize,old_vy0 + sht->bysize,0);
        sheet_refreshmap(ctl,vx0,vy0,vx0 + sht->bxsize,vy0 + sht->bysize,sht->height);
        sheet_refreshsub(ctl,old_vx0,old_vy0,old_vx0 + sht->bxsize,old_vy0 + sht->bysize,0,sht->height - 1);
        sheet_refreshsub(ctl,vx0,vy0,vx0 + sht->bxsize,vy0 + sht->bysize,sht->height,sht->height);
    }
    return;
}
```

6.3.9 sheet_slide()

- 函数声明：`void sheet_refreshmap(SHTCTL *ctl, int vx0, int vy0, int vx1, int vy1, int h0);`
- 功能：图层移动。
- 参数：图层(sht)，左上角横坐标变量(vx0)，左上角纵坐标变量(vy0)
- 返回值：空
- 调用示例：

```
void newConsole(SHTCTL *shtctl,SHEET *keyWin,int scrx,int scry){
    if(keyWin != 0){
        keywinOff(keyWin);
    }
    keyWin = openConsole(shtctl);
    sheet_slide(keyWin,CX,CY);
    sheet_updown(keyWin,shtctl->top);
    CX += 8;CY += 28;
    if(CX + CONSOLE_WIDTH >= scrx){
```

```
        CX = 8;
    }
    if(CY + CONSOLE_HEIGHT >= scry){
        CY = 8;
    }
    keywinOn(keyWin);
}
```

6.3.10 sheet_free()

- 函数声明：`void sheet_free(SHEET *sht);`
- 功能：图层释放。
- 参数：图层(sht)
- 返回值：空
- 调用示例：

```
void closeConsole(SHEET *sht){ //关闭窗口图层
    MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR;
    TASK *task = sht->task;
    memeryFree4k(memeryList,(int)sht->buf, 256 * 165);
    sheet_free(sht);
    closeConstask(task);
}
```

七、应用层API详解

> 本层API的声明，全部在`nnos.h`中。
>
> `nnos.h`为系统核心库的头文件，包含全部引导层和基础层API。对于osfun的部分，本节不再重复解析；
>
> `nnos.h`引用两个核心库，`sysdef.h`（系统常量表）和`systructual.h`（系统结构体库）。
>
> 【注意】本层全部API皆为应用级，但`Console.c`有特殊地位，之后的版本可能移入扩展层。

7.1 console.c 的API

7.1.1 总览

函数名	功能	参数	返回值	备注
consoleTask	控制台主程序	(SHEET *)sheet,(unsigned int)totalFreeSize	空	
newCMDLine	换行命令	(CONSOLE *)console	空	
sysprint	字符输出	(CONSOLE *)console,(int)charCode,(char)movFlag	空	
commandCMD	comman执行	(CONSOLE *)console,(int *)fat,(char *)command	空	
clsCMD	cls命令	(CONSOLE *)console	空	
dirCMD	dir命令	(CONSOLE *)console	空	
typeCMD	查看文件内容	(CONSOLE *)console,(int *)fat,(char *)command	空	
delCMD	删除文件	(CONSOLE *)console,(int *)fat,(char *)command	空	
sysinfoCMD	显示系统信息	(CONSOLE *)console	空	
versionCMD	显示系统版本号	(CONSOLE *)console	空	
memCMD	显示内存信息	(CONSOLE *)console	空	
runCMD	运行可执行文件	(CONSOLE *)console,(int *)fat,(char *)command	空	
exitCMD	退出控制台	(CONSOLE *)console,(int *)fat	空	
startCMD	在新窗口运行程序	(CONSOLE *)console,(char *)command	空	
syslangCMD	切换语言模式	(CONSOLE *)console,(char *)command	空	
shutdownCMD	关机	(CONSOLE *)console,(char *)command	空	待实现
invalidCMD	无效的命令	(CONSOLE *)console,(char *)command	空	
printLine	打印固定长度横线，长度为32*8像素	(CONSOLE *)console	空	
printLinex	打印指定长度横线	(CONSOLE *)console,(int)len	空	
printAddress	打印地址	(CONSOLE *)console,(unsigned char)addr	空	
sysprintl	打印一行字符串	(CONSOLE *)console,(char *)str	空	

函数名	功能	参数	返回值	备注
sysprintx	打印指定长度字符串	(CONSOLE *)console,(char *)str,(int)len	空	
sys_api	系统调用入口	(int)edi,(int)esi,(int)ebp,(int)esp,(int)ebx,(int)edx,(int)ecx,(int)eax	int	
inthandler0d0	应用程序一般异常中断处理器	(int *)esp	int	
inthandler0c	栈异常中断处理器	(int *)esp	int	

7.1.2 consoleTask()

- 函数声明：`void consoleTask(SHEET *sheet,unsigned int totalFreeSize);`
- 功能：控制台主程序。
- 参数：图层(sheet)，内存余量(totalFreeSize)
- 返回值：空
- 调用示例：

```
SHEET *openConsole(SHTCTL *shtctl){
    MEMERY_LIST *memeryList = (MEMERY_LIST *)MEMERY_ADDR;
    SHEET *sht = sheet_alloc(shtctl);
    if(sht == 0){
        sysprintl(taskNow()->console,"Create new console fail!");
        return 0;
    }
    unsigned char *buf = (unsigned char *)memeryAlloc4k(memeryList,CONSOLE_WIDTH * CONSOLE_HEIGHT);
    TASK *task = taskAlloc();
    int *cons_fifo = (int *)memeryAlloc4k(memeryList, 128 * 4);
    sheet_setbuf(sht,buf,CONSOLE_WIDTH,CONSOLE_HEIGHT,-1); //设定控制台图层

    createWindow(buf,CONSOLE_WIDTH,CONSOLE_HEIGHT,"Console",CONTROL_WINDOW,0); //创建控制台窗口
    task->consoleStack = memeryAlloc4k(memeryList, 64 * 1024); //保存控制台地址
    task->tss.esp = task->consoleStack + 64 * 1024 - 12;
    task->tss.esp = memeryAlloc4k(memeryList, 64 * 1024) + 64 * 1024 - 12;
    task->tss.eip = (int) &consoleTask;
    task->tss.es = 1 * 8;
    task->tss.cs = 2 * 8;
    task->tss.ss = 1 * 8;
    task->tss.ds = 1 * 8;
    task->tss.fs = 1 * 8;
    task->tss.gs = 1 * 8;
    *((int *) (task->tss.esp + 4)) = (int) sht;
    *((int *) (task->tss.esp + 8)) = memeryList->total_size; //地址为ESP+8
    taskRun(task, 2, 2);
    sht->task = task;
    sht->flags |= 0x20;
    initFIFOBuffer32(&task->fifo,cons_fifo,128,task);
    return sht;
```

```
}
```

7.1.3 newCMDLine()

- 函数声明：`void newCMDLine(CONSOLE *console);`
- 功能：换行命令。
- 参数：控制台(console)
- 返回值：空
- 调用示例：

```
/*版本信息*/
void versionCMD(CONSOLE *console){
    sysprintl(console,strcat("version:",NNOS_VERSION));
    newCMDLine(console);
}
```

7.1.4 sysprint()

- 函数声明：`void sysprint(CONSOLE *console,int charCode,char movFlag);`
- 功能：字符输出。
- 参数：控制台(console)，字符(charCode)，偏移(moveFlag)
- 返回值：空
- 调用示例：

```
/*打印一行字符串*/
void sysprintl(CONSOLE *console,char *str){ //打印一行，遇'\0'即停止
    //while(*str != 0){
    while(*str != 0){
        sysprint(console,*str,1);
        str++;
    }
}
```

7.1.5 commandCMD()

- 函数声明：`void commandCMD(CONSOLE *console,int *fat,char *command);`
- 功能：字符输出。
- 参数：控制台(console)，文件分配表(fat)，命令地址(command)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的consoleTask方法
//省略前驱代码
if(data == 10 + 256){ //回车键

 labelDraw(console.sheet,console.cursorX,console.cursorY,COL8_FFFFFE,COL8_000000," ",1); //输出字符
    command[console.cursorX / 8 - 2] = 0; //定位用户输入
    newCMDLine(&console);
    commandCMD(&console,fat,command);
    console.cursorX = 16;
}
//省略后继代码
```

7.1.6 clsCMD()

- 函数声明：`void commandCMD(CONSOLE *console,int *fat,char *command);`
- 功能：控制台清屏。
- 参数：控制台(console)，文件分配表(fat)，命令地址(command)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strcmp(command,"cls") == 0 || strcmp(command,"clear") == 0){ //cls,清屏
    clsCMD(console);
}
//省略后继代码
```

7.1.7 dirCMD()

- 函数声明：`void clsCMD(CONSOLE *console);`
- 功能：显示当前目录所有文件。
- 参数：控制台(console)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strcmp(command,"dir") == 0 || strcmp(command,"ls") == 0){ //dir,磁盘信息
        dirCMD(console);
}
//省略后继代码
```

7.1.8 typeCMD()

- 函数声明：`void typeCMD(CONSOLE *console,int *fat,char *command);`
- 功能：查看文件内容。
- 参数：控制台(console)，文件分配表(fat)，命令地址(command)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strncmp(command,"type ",5) == 0 || strncmp(command,"cat ",4) == 0){
    typeCMD(console,fat,command);
}
//省略后继代码
```

7.1.9 delCMD()

- 函数声明：`void delCMD(CONSOLE *console,int *fat,char *command);`
- 功能：删除文件。
- 参数：控制台(console)，文件分配表(fat)，命令地址(command)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strncmp(command, "rm ",2) == 0 || strncmp(command, "del ",2) == 0){
    delCMD(console,fat,command);
}
//省略后继代码
```

7.1.10 sysinfoCMD()

- 函数声明：`void sysinfoCMD(CONSOLE *console);`
- 功能：显示系统信息。
- 参数：控制台(console)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strcmp(command,"sysinfo") == 0 || strcmp(command,"osinfo") == 0 ||
strcmp(command,"nnos") == 0){ //sysinfo,系统信息
    sysinfoCMD(console);
}
//省略后继代码
```

7.1.11 versionCMD()

- 函数声明：`void versionCMD(CONSOLE *console);`
- 功能：显示系统版本号。
- 参数：控制台(console)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strcmp(command,"version") == 0){ //version,版本信息
    versionCMD(console);
}
//省略后继代码
```

7.1.12 memCMD()

- 函数声明：`void memCMD(CONSOLE *console);`
- 功能：显示系统信息。
- 参数：控制台(console)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strcmp(command,"mem" ) == 0 || strcmp(command,"free") == 0){ //mem,内存查询
    memCMD(console);
}
//省略后继代码
```

7.1.13 runCMD()

- 函数声明：`int runCMD(CONSOLE *console,int *fat,char *command);`
- 功能：运行可执行文件。
- 参数：控制台(console)，文件分配表(fat)，命令地址(command)
- 返回值：int(成功返回1，失败返回0)
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strncmp(command,"run ",4) == 0){
    runCMD(console,fat,command);
}
//省略后继代码
```

7.1.14 exitCMD()

- 函数声明：`void exitCMD(CONSOLE *console, int *fat);`
- 功能：显示系统信息。
- 参数：控制台(console)，文件分配表(fat)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strcmp(command,"exit") == 0){
    exitCMD(console,fat);
}
//省略后继代码
```

7.1.15 startCMD()

- 函数声明：`void startCMD(CONSOLE *console, char *command);`
- 功能：在新窗口运行程序。
- 参数：控制台(console)，命令地址(command)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strncmp(command,"start",6) == 0){
    startCMD(console,command);
}
//省略后继代码
```

7.1.16 syslangCMD()

- 函数声明：`void syslangCMD(CONSOLE *console,char *command);`
- 功能：切换语言模式。
- 参数：控制台(console)，命令地址(command)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strncmp(command, "langmode ",9) == 0) {
    syslangCMD(console,command);
}
//省略后继代码
```

7.1.17 *shutdownCMD*()

- 函数声明：`void shutdownCMD(CONSOLE *console, char *command);`
- 功能：关机。
- 参数：控制台(console)，命令地址(command)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(strcmp(command,"shutdown") == 0){
    shutdownCMD(console,command);
}
//省略后继代码
```

> 本函数尚未实现，可以调用，但关机操作无效。

7.1.18 invalidCMD()

- 函数声明：`void invalidCMD(CONSOLE *console,char *command);`
- 功能：无效的命令。
- 参数：控制台(console)，命令地址(command)
- 返回值：空
- 调用示例：

```
//完整代码见console.c的commandCMD函数
//省略前驱代码
if(command[0] != 0){ //非法输入
    if (runCMD(console,fat,command) == 0) {
        invalidCMD(console,command);
    }
}
//省略后继代码
```

7.1.19 printLine()

- 函数声明：`void printLine(CONSOLE *console);`
- 功能：打印固定长度横线，长度为32*8像素。
- 参数：控制台(console)
- 返回值：空
- 调用示例：

```
/*系统信息*/
void sysinfoCMD(CONSOLE *console){
    char info[32];
    printLine(console);
    sprintf(info, "File_System:%s\n",NNOS_FILE_SYSTEM);  //输出文件系统信息
```

```
    sysprintl(console,info);
    sprintf(info, "Version:%s\n",NNOS_VERSION);  //输出系统版本信息
    sysprintl(console,info);
    sprintf(info, "Based:%s\n",NNOS_CPU_BASED);  //输出CPU架构信息
    sysprintl(console,info);
    sprintf(info, "Campany:%s\n",NNOS_COMPANY);  //输出公司信息
    sysprintl(console,info);
    sprintf(info, "Author:%s\n",NNOS_AUTHOR);  //输出作者信息
    sysprintl(console,info);
    sprintf(info, "Email:%s\n", NNOS_EMAIL);  //输出开发者邮箱信息
    sysprintl(console,info);
    sprintf(info, "Update_Date:%s\n",NNOS_UPDATE_DATE);  //输出升级日期信息
    sysprintl(console,info);
    printLine(console);
}
```

7.1.20 printLinex()

- 函数声明：`void printLinex(CONSOLE *console,int len);`
- 功能：打印指定长度横线。
- 参数：控制台(console)，长度(像素)(len)
- 返回值：空
- 调用示例：无

7.1.21 printAddress()

- 函数声明：`void printAddress(CONSOLE *console,unsigned char addr);`
- 功能：打印地址。
- 参数：控制台(console)，地址(addr)
- 返回值：空
- 调用示例：无

7.1.22 sysprintl()

- 函数声明：`void sysprintl(CONSOLE *console,char *str);`
- 功能：打印一行字符串。
- 参数：控制台(console)，字符串(str)
- 返回值：空
- 调用示例：

```
/*打印地址*/
void printAddress(CONSOLE *console,unsigned char addr){
    char temp[32];
    sprintf(temp,"%d ",(int)addr);
    sysprintl(console,temp);
}
```

7.1.23 sysprintx()

- 函数声明：`void sysprintx(CONSOLE *console,char *str,int len);`
- 功能：打印指定长度字符串。
- 参数：控制台(console)，字符串(str)，长度(len)
- 返回值：空
- 调用示例：

```c
//完整代码见console.c的sys_api函数
//省略前驱代码
case 3:{ //接口3，打印指定长度字符串
    sysprintx(console,(char *)ebx + fileBase,ecx);break;
}
//省略后继代码
```

7.1.24 sys_api()

- 函数声明：`int *sys_api(int edi,int esi,int ebp,int esp,int ebx,int edx,int ecx,int eax);`
- 功能：系统调用入口。
- 参数：待补充。(寄存器地址，与汇编混编，无需手动调用)。
- 返回值：int(失败返回0，成功时若需返回地址等，则返回一个非零值)
- 调用示例：无

7.1.25 inthandler0d0()

- 函数声明：`int *inthandler0d0(int *esp);`
- 功能：应用程序一般异常中断处理器。
- 参数：待补充。
- 返回值：int
- 调用示例：无

7.1.26 inthandler0c()

- 函数声明：`int *inthandler0c(int *esp);`
- 功能：栈异常中断处理器。
- 参数：待补充
- 返回值：空
- 调用示例：无

7.2 service.c的API

7.2.1 总览

函数名	功能	参数	返回值	备注
tssBMain	测试程序	(SHEET *)sht_win_b	空	

7.2.2 tssBMain()

> 本节API无用，不建议使用，之后的版本中将移除。

- 函数声明：`void tssBMain(SHEET *sht_win_b);`
- 功能：测试程序。
- 参数：图层(sht_win_b)
- 返回值：空
- 调用示例：无

7.3 string.c的API

> 本节API皆不建议使用，之后的版本中将移动或移除。

7.3.1 总览

函数名	功能	参数	返回值	备注
convertToUppercase	转为大写字母	(char *)str	空	
convertToLowercase	转为小写字母	(char *)str	空	

7.3.2 convertToUppercase()

- 函数声明：`void convertToUppercase(char *str);`
- 功能：转为大写字母。
- 参数：字符串(str)
- 返回值：空
- 调用示例：无

7.3.3 convertToLowercase()

- 函数声明：`void convertToLowercase(char *str);`
- 功能：转为小写字母。
- 参数：字符串(str)
- 返回值：空
- 调用示例：无

八、系统调用详解

> 本层API的声明，全部在`api.h`中。
>
> `api.h`为系统调用的头文件，包含全部系统调用API。
>
> 系统调用提供给应用程序开发者，以开发应用程序。
>
> 【注意】本层全部API皆为应用级。

8.1 系统调用常量

常量名	值	功能	备注
API_VERSION	"0.34d"	API版本	值随系统版本修改
COLOR_BASE	27	颜色库数目	值随系统升级增减

8.2 字符输出API

8.2.1 总览

函数名	功能	声明	备注
api_printc	输出单个字符	api_printc(int c);	
api_printl	绘制字符串	api_printl(char *s);	

8.2.2 api_printc()

- 函数声明：`void api_printc(int c);`
- 功能：输出单个字符。
- 参数：字符(c)
- 返回值：空
- 调用示例：

```
/**
 * File name:hello03.c
**/
void api_printc(int c);
void api_return();

void NNOSMain(){
    api_printc('H');
    api_printc('e');
    api_printc('l');
    api_printc('l');
    api_printc('o');
    api_printc(' ');
    api_printc('w');
    api_printc('o');
    api_printc('r');
    api_printc('l');
    api_printc('d');
    api_printc(',');
    api_printc('n');
    api_printc('n');
    api_printc('o');
    api_printc('s');
    api_printc('!');
    api_return();
}
```

8.2.3 api_printl()

- 函数声明：`void api_printl(char *s);`
- 功能：绘制字符串。
- 参数：字符串(s)
- 返回值：空
- 调用示例：

```
/**
 * File name:hello04.c
**/
void api_printl(char *s);
void api_return();

void NNOSMain(){
    api_printl("Hello world, nnos API.");
    api_return();
}
```

8.3 图形绘制API

8.3.1 总览

函数名	功能	声明	备注
api_point	画点	void api_point(int win,int x,int y,int vcolor);	
api_linewin	画线	void api_linewin(int win,int x0,int y0,int x1,int y1, int vcolor);	
api_boxwin	画色块（兼容）	void api_boxwin(int win,int x0,int y0,int x1,int y1,int vcolor);	

8.3.2 api_point()

- 函数声明：`void api_point(int win,int x,int y,int vcolor);`
- 功能：画点。
- 参数：窗口句柄(win)，点横坐标(x)，点纵坐标(y)，点颜色(vcolorf)
- 返回值：空
- 调用示例：

```
/**
 * File name:point02.c
**/
#include "api.h"

#define WIDTH 150
#define HEIGHT 100

int rand();

void NNOSMain()
{
    char *buffer;
    int win,i,x,y;
    api_initmalloc();
    buffer = api_malloc(WIDTH * HEIGHT);
    win = api_window(buffer,WIDTH,HEIGHT,-1,"Points");
    api_boxwin(win,6,26,143,93,0);
    for (i = 0; i < 50; i++) {
        x = (rand() % 137) + 6;
        y = (rand() %  67) + 26;
        api_point(win + 1,x,y,4); //窗口句柄偏移1，变为奇数，不刷新图层
    }
    api_refreshwin(win,6,26,144,94); //统一刷新图层
    while(api_getkey(1) != 0x0a);
    api_return();
}
```

8.3.3 api_linewin()

- 函数声明：`void api_linewin(int win,int x0,int y0,int x1,int y1, int vcolor);`
- 功能：画线。
- 参数：窗口句柄(win)，A端横坐标(x0)，A端纵坐标(y0)，B端横坐标(x1)，B端纵坐标(y1)，颜色(vcolor)
- 返回值：空
- 调用示例：

```
/**
 * File name:lines.c
**/
#include "api.h"

#define WIDTH 160
#define HEIGHT 100

void NNOSMain(){
    char *buffer;
    int window, i;
    api_initmalloc();
    buffer = api_malloc(WIDTH * HEIGHT);
    window = api_window(buffer,160,100,-1,"Lines");
    for(i = 0;i < 8;i++){
        api_linewin(window+1,8,26,77,i * 9 + 26,i);
        api_linewin(window+1,88,26,i * 9 + 88,89,i);
    }
    api_refreshwin(window,6,26,154,90);
    while(api_getkey(1) != 0x0a);
    api_closewin(window);
    api_return();
}

```

8.3.4 api_boxwin()

- 函数声明：`void api_boxwin(int win,int x0,int y0,int x1,int y1,int vcolor);`
- 功能：画色块（兼容）。
- 参数：窗口句柄(win)，色块左上角横坐标(x0)，色块右下角横坐标(x1)，色块左上角纵坐标(y0)，色块右下角纵坐标(y1)，色块颜色(vcolor)
- 返回值：空
- 调用示例：

```
/**
 * File name:ball.c
**/
#include "api.h"

void NNOSMain(void)
{
    int win, i, j, dis;
    char buf[216 * 237];
    struct POINT {
        int x, y;
    };

```

```
    static struct POINT table[16] = {
        { 204, 129 }, { 195,  90 }, { 172,  58 }, { 137,  38 }, {  98,  34 },
        {  61,  46 }, {  31,  73 }, {  15, 110 }, {  15, 148 }, {  31, 185 },
        {  61, 212 }, {  98, 224 }, { 137, 220 }, { 172, 200 }, { 195, 168 },
        { 204, 129 }
    };

    win = api_window(buf, 216, 237, -1, "bball");
    api_boxwin(win, 8, 29, 207, 228, 0);
    for (i = 0; i <= 14; i++) {
        for (j = i + 1; j <= 15; j++) {
            dis = j - i;
            if (dis >= 8) {
                dis = 15 - dis;
            }
            if (dis != 0) {
                api_linewin(win, table[i].x, table[i].y, table[j].x, table[j].y, 8 - dis);
            }
        }
    }

    for (;;) {
        if (api_getkey(1) == 0x0a) {
            break;
        }
    }
    api_return();
}
```

8.4 窗口相关API

8.4.1 总览

函数名	功能	声明	备注
api_window	窗口创建	int api_window(char *buffer,int width,int height,int vcolor,char *caption);	
api_ascwin	在窗口上绘制字符串	void api_ascwin(int win,int x,int y,int vcolor,int len,char *str);	
api_refreshwin	画线	api_refreshwin(int win, int x0, int y0, int x1, int y1);	
api_closewin	窗口刷新	api_closewin(int win);	

8.4.2 api_window()

- 函数声明：`int api_window(char *buffer,int width,int height,int vcolor,char *caption);`
- 功能：窗口创建。
- 参数：缓冲区(buffer)，宽度(width)，高度(height)，颜色(vcolor)，标题(caption)

- 返回值：int(窗口句柄)
- 调用示例：

```
/**
 * File name:helwin04.c
**/
#define WIDTH 150
#define HEIGHT 50

#include "api.h"

void NNOSMain(){
    char buffer[WIDTH * HEIGHT];
    int window;
    api_initmalloc();
    window = api_window(buffer,WIDTH,HEIGHT, -1,"API_WIN");
    api_boxwin(window,8,24,141,42,1);
    api_ascwin(window,28,28,4,12,"Hello world!");
    while(api_getkey(1) != 0x0a);
    api_return();
}
```

8.4.3 api_ascwin()

- 函数声明：`void api_ascwin(int win,int x,int y,int vcolor,int len,char *str);`
- 功能：在窗口上绘制字符串。
- 参数：缓冲区(buffer)，宽度(width)，高度(height)，颜色(vcolor)，标题(caption)
- 返回值：空
- 调用示例：

> 参见api_window()之调用示例。

8.4.4 api_refreshwin()

- 函数声明：`void api_refreshwin(int win, int x0, int y0, int x1, int y1);`
- 功能：窗口刷新。
- 参数：窗口句柄(win)，左上角横坐标(x0)，左上角纵坐标(y0)，右下角横坐标(x1)，右下角纵坐标(y1)
- 返回值：空
- 调用示例：

```
/**
 * File name:color02.c
**/
#include "api.h"

void NNOSMain(){
    char *buf;
    int win, x, y;
    api_initmalloc();
    buf = api_malloc(144 * 164);
    win = api_window(buf, 144, 164, -1, "Color2");
    for (y = 0; y < 128; y++) {
        for (x = 0; x < 128; x++) {
            buf[(x + 8) + (y + 28) * 144] = rgb2pal(x * 2, y * 2, 0, x, y);
        }
```

```
    }
    api_refreshwin(win, 8, 28, 136, 156);
    api_getkey(1);
    api_return();
}

unsigned char rgb2pal(int r, int g, int b, int x, int y){
    static int table[4] = { 3, 1, 0, 2 };
    int i;
    x &= 1;
    y &= 1;
    i = table[x + y * 2];
    r = (r * 21) / 256;
    g = (g * 21) / 256;
    b = (b * 21) / 256;
    r = (r + i) / 4;
    g = (g + i) / 4;
    b = (b + i) / 4;
    return COLOR_BASE + r + g * 6 + b * 36;
}
```

8.4.5 api_closewin()

- 函数声明：`void api_closewin(int win);`
- 功能：关闭窗口。
- 参数：窗口句柄(win)
- 返回值：空
- 调用示例：

```
/**
 * File name:snake.c
**/
#include "api.h"

#define WIDTH 160
#define HEIGHT 100
#define DE_X WIDTH /2
#define DE_Y HEIGHT / 2 + 10
#define FOOD_NUM 21
#define LINE 5
#define CAPTION 20
#define LINE_CAPTION LINE+CAPTION
#define STEP 8
#define WORD_WIDTH 8
#define WORD_HEIGHT 16
#define DE_LEN 1

typedef struct FOOD{
    int x;
    int y;
    int color;
    int live;
}FOOD;

int rand();

void NNOSMain(){
```

```
    char *buf;
    int window,i,x,y,color;
    int flag = 1;
    api_initmalloc();
    buf = api_malloc(WIDTH * HEIGHT);
    window = api_window(buf,WIDTH,HEIGHT,-1,"Snake");
    api_boxwin(window,LINE,LINE_CAPTION,WIDTH - 5,HEIGHT - 5,0);
    for (i = 0; i < FOOD_NUM; i++) {
        x = (rand() % WIDTH - 18) + 6;
        y = (rand() %  HEIGHT - 28) + 26;
        color = (rand() %  25) + 1;
        api_point(window,x,y,color);
    }
    x = DE_X;y = DE_Y;

    api_ascwin(window,x,y,4,DE_LEN,"*");
    while(1 && (flag == 1)){
        i = api_getkey(1);
        api_ascwin(window,x,y,0,1,"*");
        switch(i){
            case '4':{
                if(x > LINE + 6){
                    x -= STEP;
                }
                break;
            }
            case '6':{
                if(x < WIDTH - WORD_WIDTH - 8){
                    x += STEP;
                }
                break;
            }
            case '8':{
                if(y > LINE_CAPTION + 1){
                    y -= STEP;
                }
                break;
            }
            case '2':{
                if(y < HEIGHT - WORD_HEIGHT - LINE - 2){
                    y += STEP;
                }
                break;
            }
            case 0x0a:{
                flag = 0;
                break;
            }
        }
        api_ascwin(window,x,y,4,1,"*");
    }
    api_closewin(window);
    api_return();
}
```

8.5 内存相关API

8.5.1 总览

函数名	功能	声明	备注
api_initmalloc	初始化内存	void api_initmalloc();	
api_malloc	申请内存	char *api_malloc(int size);	
api_free	释放内存	void api_free(char *addr, int size);	

8.5.2 api_initmalloc()

- 函数声明：`void api_initmalloc();`
- 功能：初始化内存。
- 参数：空
- 返回值：int(窗口句柄)
- 调用示例：

> 参看api_closewin()节之调用示例。

8.5.3 api_malloc()

- 函数声明：`char *api_malloc(int size);`
- 功能：申请内存。
- 参数：申请内存大小(size)
- 返回值：char(首地址)
- 调用示例：

> 参看api_closewin()节之调用示例。

8.5.4 api_free()

- 函数声明：`void api_free(char *addr, int size);`
- 功能：申请内存。
- 参数：首地址(addr)，大小(size)
- 返回值：空
- 调用示例：无

> 在调用窗口相关API时，系统会自动完成内存释放，无需手动管理。
>
> 其他情况待定。

8.6 控制相关API

8.6.1 总览

函数名	功能	声明	备注
api_return	返回调用者	void api_return();	

8.6.2 api_return()

- 函数声明：`void api_return();`
- 功能：返回调用者。
- 参数：空
- 返回值：空
- 调用示例：无

> 参看api_boxwin()之示例。

8.7 输入相关API

8.7.1 总览

函数名	功能	声明	备注
api_getkey	接收键盘输入	int api_getkey(int mode);	

8.7.2 api_getkey()

- 函数声明：`int api_getkey(int mode);`
- 功能：接收键盘输入。
- 参数：模式(mode)
- 返回值：int(字符的ASCII码)
- 调用示例：

> 参看api_closewin()之示例。
>
> `mode`值通常置为`1`。

8.8 定时器相关API

8.8.1 总览

函数名	功能	声明	备注
api_alloctimer	申请定时器	int api_alloctimer();	
api_inittimer	初始化定时器	void api_inittimer(int timer, int data);	
api_settimer	设定定时器	void api_settimer(int timer, int time);	
api_freetimer	释放定时器	void api_freetimer(int timer);	

8.8.2 api_alloctimer()

- 函数声明：`int api_alloctimer();`
- 功能：申请定时器。
- 参数：空
- 返回值：int(定时器句柄)
- 调用示例：

```
/**
 * File name:timers.c
**/
#include <stdio.h>
```

```c
#include "api.h"
#define WIDTH 150
#define HEIGHT 80

void NNOSMain()
{
    char *buf, s[12];
    int win, timer, sec = 0, min = 0, hou = 0;
    api_initmalloc();
    buf = api_malloc(150 * 50);
    win = api_window(buf, 150, 50, -1, "Timers");
    timer = api_alloctimer();
    api_inittimer(timer, 128);
    while(1){
        sprintf(s, "%5d:%02d:%02d", hou, min, sec);
        api_boxwin(win, 28, 27, 115, 41, 7);
        api_ascwin(win, 28, 27, 0, 11, s);
        api_settimer(timer, 100);
        if (api_getkey(1) != 128) {
            break;
        }
        sec++;
        if (sec == 60) {
            sec = 0;
            min++;
            if (min == 60) {
                min = 0;
                hou++;
            }
        }
    }
    api_return();
}
```

8.8.3 api_inittimer()

- 函数声明：`void api_inittimer(int timer, int data);`
- 功能：初始化定时器。
- 参数：定时器句柄(timer)，定时器数据(data)
- 返回值：空
- 调用示例：

> 参看api_alloctimer()之示例。

8.8.4 api_settimer()

- 函数声明：`void api_settimer(int timer, int time);`
- 功能：设定定时器。
- 参数：定时器句柄(timer)，超时时间(毫秒)(time)
- 返回值：空
- 调用示例：

> 参看api_alloctimer()之示例。

8.8.5 api_freetimer()

- 函数声明：`void api_freetimer(int timer);`
- 功能：释放定时器。
- 参数：定时器句柄(timer)
- 返回值：空
- 调用示例：无

8.9 蜂鸣发声器相关API

8.9.1 总览

函数名	功能	声明	备注
api_beep	调用蜂鸣发声器	void api_beep(int tone);	

8.9.2 api_beep()

- 函数声明：`void api_beep(int tone);`
- 功能：调用蜂鸣发声器。
- 参数：音调(tone)
- 返回值：空
- 调用示例：

```
/**
 * File name:beep02.c
**/
#include "api.h"

void NNOSMain(){
    int i, timer;
    timer = api_alloctimer();
    api_inittimer(timer, 128);
    for (i = 20000; i <= 20000000; i += i / 100) { //20KHz~20Hz，人可听到的声音范围
        api_beep(i); //i以1%的速度递增
        api_settimer(timer, 1); //每次发声间隔0.01秒
        if (api_getkey(1) != 128) {
            break;
        }
    }
    api_beep(0);
    api_return();
}
```

> 参数`tone`，音调，值越大音调越高，实质为频率，超出人耳听力范围(`16Hz~24000Hz`)将表现为无声。

8.10 定时器相关API

8.10.1 总览

函数名	功能	声明	备注
api_fopen	打开文件	int api_fopen(char *fileName);	
api_fclose	关闭文件	void api_fclose(int fileHanle);	
api_fseek	定位文件	void api_fseek(int fileHanle, int offset, int mode);	
api_fsize	获取文件大小	void api_freetimer(int timer);	
api_fread	读取文件	int api_fread(char *buffer, int maxsize, int fileHanle);	

8.10.2 api_fopen()

- 函数声明：`int api_fopen(char *fileName);`
- 功能：打开文件。
- 参数：文件句柄(fileHandel)
- 返回值：空
- 调用示例：

> 参看`根目录/app`下的原文件[notepad.c]。

8.10.3 api_fclose()

- 函数声明：`void api_fclose(int fileHanle);`
- 功能：关闭文件。
- 参数：文件句柄(fileHandel)
- 返回值：空
- 调用示例：

> 参看`根目录/app`下的原文件[notepad.c]。

8.10.4 api_fseek()

- 函数声明：`void api_fseek(int fileHanle, int offset, int mode);`
- 功能：定位文件。
- 参数：文件句柄(fileHandel)，偏移量(offset)，模式(mode)
- 返回值：空
- 调用示例：

> 参看`根目录/app`下的原文件[notepad.c]。

8.10.5 api_fclose()

- 函数声明：`int api_fsize(int fileHanle, int mode);`
- 功能：获取文件大小。
- 参数：文件句柄(fileHandel)，模式(mode)
- 返回值：int(文件大小)
- 调用示例：

> 参看`根目录/app`下的原文件[notepad.c]。

8.10.6 api_fread()

- 函数声明：`int api_fread(char *buffer, int maxsize, int fileHanle);`
- 功能：读取文件。
- 参数：缓冲区(buffer)，最大大小(maxsize)，文件句柄(fileHandle)
- 返回值：int(文件句柄)
- 调用示例：

> 参看`根目录/app`下的原文件[notepad.c]。

8.11 控制台相关API

8.11.1 总览

函数名	功能	声明	备注
api_command	获取控制台输入	int api_command(char *buffer, int maxSize);	

8.11.2 api_command()

- 函数声明：`int api_command(char *buffer, int maxSize);`
- 功能：获取控制台输入。
- 参数：缓冲区(buffer)，最大大小(maxsize)
- 返回值：int(字符的ASCII码)
- 调用示例：

> 参看`根目录/app`下的原文件[caltor.c]。

8.13 字体相关API

8.13.1 总览

函数名	功能	声明	备注
api_getlang	获取当前字库地址	int api_getlang();	

8.13.2 api_getlang()

- 函数声明：`int api_getlang();`
- 功能：获取当前字库地址。
- 参数：空
- 返回值：int(字库地址)
- 调用示例：无

九、C语言标准函数库

> 本节API的声明，暂存在`api.h`中。
>
> `api.h`为系统调用的头文件，包含全部系统调用API。
>
> 系统调用提供给应用程序开发者，以开发应用程序。
>
> 【注意】本节全部API皆为应用级。
>
> 标准函数库尚未完全实现。

9.1 已实现的C标准函数

9.1.1 总览

函数名	功能	声明	备注
putchar	输入一个字符	int putchar(int c);	
exit	退出	int exit(int status);	
printf	标准输出	int printf(char *format,...);	
malloc	申请内存	void *malloc(int size);	
free	释放内存	void free(void *p);	
scanf	标准输入	void scanf(char *format,...);	尚存问题

> C标准函数不再提供示例。
>
> 标准函数之实现达到一定规模后，将单独建立文档，分类给出详解。
>
> `scanf()`**尚存问题，暂勿使用。**

十、系统结构体简表

> 系统结构体全部存放于`根目录/lib/systructual.h`中。

10.1 总览

结构体名	功能	类别	备注
SEGMENT_DESCRIPTOR	全局描述符(GDT)	描述符	
GATE_DESCRIPTOR	中断描述符(IDT)	~	
FIFO_BUFFER	鼠标键盘专用缓冲区	缓冲区	
FIFO_BUFFER32	32位通用缓冲区	~	
TASK	任务结构体	任务	
TASK_LIST	任务列表结构体	~	
TASK_LEVEL	任务优先级队列	~	
TSS32	任务状态段(TSS)	~	
BOOT_INFO	BOOT信息	系统硬件描述	
SCREEN	屏幕结构体	~	
MOUSE_CURSOR	鼠标指针结构体	~	
FREE_BLOCK	内存空闲块	内存	
MEMERY_LIST	内存空闲块表结构体	~	
TIMER	定时器结构体	定时器	
TIMER_LIST	定时器列表结构体	~	
PIX_POINT	像素点结构体	绘图	
BOX	色块结构体	~	
PICTURE	图形结构体	~	
WINDOW	窗口结构体	~	
WINDOW_LIST	窗口列表结构体	~	
COVER	图层结构体	图层	
COVER_LIST	图层列表结构体	~	
SHEET	图层结构体（第三方）	~	
SHTCTL	图层列表结构体（第三方）	~	
FILE_INFO	文件信息	文件	
FILE_HANDLE	文件处理器	~	
CONSOLE	控制台	控制台	

10.2 描述符

10.2.1 SEGMENT_DESCRIPTOR()

```
/*全局描述符(GDT)*/
typedef struct SEGMENT_DESCRIPTOR{  //以CPU信息为基础的结构体，存放8字节内容（CPU规格要求，段信息按8字节写入内存）
    short LOW_LIMIT;                //低位上限
    short LOW_BASE;                 //低位基址（低位2字节基地址）
    char MID_BASE;                  //中位基址（中位1字节基地址）
    char ACCESS_PER;                //操作权限（禁止写入、禁止执行、系统专用）（0x9a，系统模式；0x92，应用模式）
    char HIGH_LIMIT;                //高位上限
    char HIGH_BASE;                 //高位基址（高位1字节基地址，低、中、高三个基地址共32位,分三段，可兼容80286）
}SEGMENT_DESCRIPTOR;
```

10.2.2 GATE_DESCRIPTO()

```
/*中断描述符(IDT)*/
typedef struct GATE_DESCRIPTOR{     //以CPU信息为基础的结构体，存放8字节内容
    short LOW_OFFSET;               //低位偏移
    short SELECTOR;                 //段选择子（段号）
    char DW_COUNT;                  //计数器
    char ACCESS_PER;                //段访问权限
    short HIGH_OFFSET;              //高位偏移
}GATE_DESCRIPTOR;
```

> 1.段上限最大为4GB，为32位保护模式可访问的最大内存。段本身4字节，4字节，共需8字节，因此段上限只能用20位，最大1MB。Gbit=1,32位模式，Gbit=0,16位模式；
>
> 2.段中设有标志位Gbit，标志位为1时limit单位不解释为byte,而解释为页，1page=4KB。4KB*1M = 4GB。Gbit=granularity,单位大小；
>
> 3.ACCESS_PER为访问权限，共12位，高4位放在HIGH_LIMT中，xxxx0000xxxxxxxx.高4位为扩展访问权，386以后才使用由GD00构成，Gbit，段模式。
>
> 4.ACCESS_PER低8位构成（80386前便有）
>
> 00000000(0x00)：未使用的记录（descriptor table）
>
> 10010010(0x92)：系统专用，可读可写段，不可执行。
>
> 10011010(0x9a)：系统专用，可执行段，可读不可写。
>
> 11110010(0xf2)：应用程序用，可读写段，不可执行。
>
> 11111010(0xfa)：应用程序用，可执行段，可读不可写。
>
> 5.32位模式下，CUP有系统模式(内核模式，ring0，环节0)和应用模式(用户模式,ring3)之分。此外，ring1、ring2由设备驱动器(device driver)等使用。

10.3 缓冲区

10.3.1 FIFO_BUFFER()

```
/*鼠标键盘专用缓冲区*/
typedef struct FIFO_BUFFER{
    unsigned char *buffer; //缓冲区首地址
    int next_write; //写指针
    int next_read; //读指针
    int size; //总容量
    int free; //空闲容量
    int flags; //标志位
}FIFO_BUFFER;
```

10.3.2 **FIFO_BUFFER32**()

```
/*32位通用缓冲区*/
typedef struct FIFO_BUFFER32{
    int *buffer; //缓冲区首地址
    int next_write; //写指针
    int next_read; //读指针
    int size; //总容量
    int free; //空闲容量
    int flags; //标志位
    struct TASK *task; //任务指针
}FIFO_BUFFER32;
```

10.4 任务

10.4.1 TASK()

```
/*任务结构体*/
typedef struct TASK{
    int selector; //段选择子(段选择符，段地址标识，段编号)
    int flag; //状态标志
    TSS32 tss; //任务状态段
    int priority; //优先级
    int level; //优先级队列
    struct FIFO_BUFFER32 fifo; //32位通用缓冲区
    struct CONSOLE *console; //控制台指针
    int base; //数据段首地址
    int consoleStack; //控制台地址栈，用于记录打开的控制台的地址，关闭时使用
    struct SEGMENT_DESCRIPTOR ldt[2]; //LDT段
    struct FILE_HANDLE *file_handle; //文件处理器
    int *fat; //文件分配表指针
    char *command;
    char lang_mode; //语言模式
    int lang_byte; //接收到全角时存放第一个字节内容，接收到半角或全角完成后此变量被置为
0
}TASK;
```

10.4.2 TASK_LIST()

```
/*任务列表结构体*/
typedef struct TASK_LIST{
    int level_run; //运行中的任务数量
    char level_change;
    //int now; //当前正在运行的任务
    TASK_LEVEL task_level[LEVEL_MAX]; //任务栏运行队列
    TASK tasks0[TASK_MAX]; //任务列表
}TASK_LIST;
```

10.4.3 TASK_LEVEL()

```
/*任务优先级队列*/
typedef struct TASK_LEVEL{
    int number; //运行中的任务数量
    int now; //当前正在运行的任务
    TASK *tasks[TASK_LEVEL_MAX]; //优先级队列任务列表
}TASK_LEVEL;
```

10.4.4 TSS32()

```
/*任务状态段*/
typedef struct TSS32{ //26个int成员,104字节
    int backlink,esp0,ss0,esp1,ss1,esp2,ss2,cr3;  //与任务设置有关的信息，可暂时忽略
    int eip,eflags,eax,ecx,edx,ebx,esp,ebp,esi,edi; //32位寄存器
    int es,cs,ss,ds,fs,gs; //16位寄存器
    int ldtr,iomap; //任务设置相关，任务奇幻史不会被CPUT写入，不可忽略，必须按规定赋值
}TSS32;
```

10.5 系统硬件描述

10.5.1 BOOT_INFO()

```
/*BOOT信息结构体*/            //信息数据来自于syshead.asm
 typedef struct BOOT_INFO{    //指针指向相应信息
    char cyls_max;           //磁盘最大装载柱面号（启动区读硬盘读到何处为止）
    char leds_status;        //键盘LED指示灯状态
    char vram_mode;           //显卡模式（位数，颜色模式）
    char reserve;             //保留区域
    short screen_width;      //显示器行分辨率（单位：像素）
    short screen_height;     //显示器列分辨率（单位：像素）
    char *vram_base;           //显存首地址
}BOOT_INFO;
```

10.5.2 SCREEN()

```c
/*屏幕结构体*/
typedef struct SCREEN{  //【害群之马，永不复用】
    unsigned char *vram;
    short scrx;
    short scry;
    int centerX;
    int centerY;
}SCREEN;
```

10.5.3 MOUSE_CURSOR()

```c
/*鼠标指针结构体*/
typedef struct MOUSE_CURSOR{
    unsigned char dataBuffer[CURSOR_DATA_SIZE];
    unsigned phase;
    int mx;
    int my;
    int mbutton;
    char cursor_graph[CURSOR_GRAPH_SIZE];
}MOUSE_CURSOR;
```

10.6 内存

10.6.1 FREE_BLOCK()

```c
/*内存空闲块结构体*/
typedef struct FREE_BLOCK{ //空闲块
    unsigned int addr;
    unsigned int size;
}FREE_BLOCK;
```

10.6.2 MEMERY_LIST()

```c
/*内存空闲块表结构体*/
typedef struct MEMERY_LIST{ //空闲块表
    int number; //空闲块数量
    int max_number; //最大空闲内存块数量
    int lost_number; //释放内存失败的内存块数量
    int lost_size;  //释放内存失败的次数
    unsigned int total_size;
    unsigned int total_free_size; //空闲内存总大小
    FREE_BLOCK free[MEMERY_LIST_SIZE]; //内存表主体
}MEMERY_LIST;
```

10.7 定时器

10.7.1 TIMER()

```c
/*定时器结构体*/
typedef struct TIMER{
    struct TIMER *next;
    unsigned int timeout; //超时时间
    int data; //超时信息
    FIFO_BUFFER32 *fifo; //信息缓冲区
    unsigned int flag; //状态信息(未使用、已设置<使用>、运行中)
    unsigned int flag2; //自动关闭标志，是否在程序结束后自动关闭
}TIMER;
```

10.7.2 TIMER_LIST()

```c
/*定时器列表结构体*/
typedef struct TIMER_LIST{
    unsigned int count; //计时器
    unsigned int next; //下一个即将超时的时刻（可改进为下一个计时器编号）
    TIMER *timers; //运行中的定时器链表
    TIMER timer[TIMER_MAX]; //定时器数组
}TIMER_LIST;
```

10.8 定时器

10.8.1 PIX_POINT()

```c
/*像素点结构体*/
typedef struct PIX_POINT{
    int px;                 //点横坐标
    int py;                 //点纵坐标
    unsigned char vcolor;   //填充色
}PIX_POINT;
```

10.8.2 BOX()

```c
/*色块结构体*/
typedef struct BOX{
    int bx0;                //矩形横向起点
    int by0;                //矩形纵向起点
    int bx1;                //矩形横向终点
    int by1;                //矩形纵向终点
    unsigned char vcolor;   //填充色
    char box_name[8];
}BOX;
```

10.8.3 PICTURE()

```c
/*图形结构体*/
typedef struct PICTURE{
    int px;               //横坐标
    int py;               //纵坐标
    int width;           //图形宽度
    int height;          //图形高度
    //int size;            //每行像素数
    char *base;          //图形内存基址
}PICTURE;
```

10.8.4 WINDOW()

```c
/*窗口结构体*/
typedef struct WINDOW{
    int type; //窗口类型（1-普通，2-提示，3-询问，4全屏）
    int X;
    int Y;
    int width;
    int height;
    char *caption;
    unsigned char foreColor;
    unsigned char backColor;
    int visible;
    int lock;
}WINDOW;
```

10.8.5 WINDOW_LIST()

```c
/*窗口列表结构体*/
typedef struct WINDOW_LIST{
    int number;
    int now;
    WINDOW autoWindow[100];
}WINDOW_LIST;
```

10.9 图层

10.9.1 COVER()

```c
/*图层结构体*/
typedef struct COVER{
    int id; //图层ID
    PICTURE picture;  //图层图形数组（宽、高、横纵坐标、图形首地址）
    int order; //图层层级
    int flag; //使用状态
    int trans_color; //透明色号
    struct COVER *next;
}COVER;
```

10.9.2 COVER_LIST()

```
/*图层列表结构体*/
typedef struct COVER_LIST{
    SCREEN screen;  //屏幕信息（宽、高、显存首地址）
    //COVER covers[COVER_LIST_SIZE]; //图层数组
    COVER *head; //图层链表首指针
    COVER *rear; //图层链表尾指针
    int top; //当前最高图层层次
    int id_max; //当前最大id
}COVER_LIST;
```

10.9.3 SHEET()

```
/*图层结构体（第三方）*/
typedef struct SHEET{
    unsigned char *buf;
    int bxsize, bysize, vx0, vy0, col_inv, height, flags;
    struct SHTCTL *ctl;
    TASK *task;
}SHEET;
```

10.9.4 SHTCTL()

```
/*图层列表结构体（第三方）*/
typedef struct SHTCTL{
    unsigned char *vram, *map;
    int xsize, ysize, top;
    struct SHEET *sheets[MAX_SHEETS];
    struct SHEET sheets0[MAX_SHEETS];
}SHTCTL;
```

10.10 文件

10.10.1 FILE_INFO()

```
/*文件信息*/
typedef struct FILE_INFO{ //根据nnos.img二进制文件0x002600字节之后内容创建
    unsigned char name[8];  //文件名,字符编码信息无负数
    unsigned char exten[3]; //拓展名
    unsigned char type;     //文件类型（常为0x00或0x20），0x01 只读，0x02 隐藏，0x04 系统文件，0x08 非文件信息（磁盘名称等），0x10 目录，多个属性只需相加
    char reserve[10];       //保留区域，FAT12格式保留区域，微软规定，自行定义易发生兼容问题
    unsigned short time;    //修改时间，须公式转换，下同
    unsigned short date;    //修改日期
    unsigned short cluster; //簇号，软盘每簇为1个扇区。此处标识文件从哪个扇区开始存放，小端模式
    unsigned int size;      //文件大小
}FILE_INFO;
```

10.10.2 FILE_HANDLE()

```
/*文件处理器*/
typedef struct FILE_HANDLE{
    char *buffer;
    int size;
    int pos;
}FILE_HANDLE;
```

10.11 控制台

10.11.1 CONSOLE()

```
/*控制台*/
typedef struct CONSOLE{
    SHEET *sheet;     //控制台图层指针
    int cursorX; //光标横坐标
    int cursorY; //光标纵坐标
    int cursorC; //光标颜色
    TIMER *timer;
}CONSOLE;
```

> 若文件名第一个字节为0xe5，代表此文件已被删除
>
> 第一个字节为0x00，代表此段不任何文件信息
>
> 从磁盘镜像0x004200开始存放nnos.sys，文件信息最多存放224个
>
> 文件名不够8字节或无拓展名皆用空格补足，皆为大写字母
>
> 文件属性：
>
> 0x01 只读
>
> 0x02 隐藏
>
> 0x04 系统文件
>
> 0x08 非文件信息（磁盘名称等）
>
> 0x10 目录
>
> 多个属性只需相加，只读+隐藏=0x01+0x03 = 0x04

十一、系统常量简表

> 系统结构体全部存放于`根目录/lib/sysdef.h`中。

11.1 版本号

常量名	值	功能	备注
NNOS_VERSION	"NNOS 0.34d"	系统版本号	值随系统版本修改

11.2 系统信息

常量名	值	功能	备注
NNOS_FILE_SYSTEM	"FAT12"	文件系统信息	
NNOS_CPU_BASED	"Intel_80x86"	CPU架构	
NNOS_COMPANY	"NNRJ"	公司	随时修改
NNOS_AUTHOR	"Liu Dongxu"	作者	随时修改
NNOS_EMAIL	"tianhehechu@qq.com"	开发者邮箱	随时修改
NNOS_UPDATE_DATE	"2019-5-1"	更新日期	值随系统升级更改

11.3 BOOT信息

常量名	值	功能	备注
BOOTINFO_ADR	0x00000ff0	BOOT信息首地址	

11.4 GDT&IDT

常量名	值	功能	备注
GDT_ADR	0x00270000	GDT基址	
IDT_ADR	0x0026f800	IDT基址	
GDT_LIMIT	0x0000ffff	GDT上限	
IDT_LIMIT	0x000007ff	IDT上限	
BOOT_ADR	0x00280000	BOOT首地址	
BOOT_LIMIT	0X0007ffff	BOOT上限	
DATE32_RW_PRE	0x4092	读写权限，应用模式	（用户模式，用户态）
CODE32_ER_PRE	0x409a	运行权限，系统模式	（内核模式，内核态）
INTGATE32_PRE	0x008e	允许中断	
GDT_SELECTOR_MAX	8192	GDT选择子最大值	
TSS32_PRE	0x0089	多任务	
LDT_PRE	0x0082	LDT段访问权限	

11.5 显卡常量

常量名	值	功能	备注
COLORNUM	27	颜色库数目	

11.6 PIC常量

常量名	值	功能	备注
PIC0_ICW1	0x0020	/	可编程中断处理器
PIC0_ICW2	0x0021	/	
PIC0_ICW3	0x0021	/	
PIC0_ICW4	0x0021	/	
PIC0_OCW2	0x0020	/	
PIC0_IMR	0x0021	/	
PIC1_ICW1	0x00a0	/	
PIC1_ICW2	0x00a1	/	
PIC1_ICW3	0x00a1	/	
PIC1_ICW4	0x00a1	/	
PIC1_OCW2	0x00a0	/	
PIC1_IMR	0x00a1	/	

11.7 PIT常量

常量名	值	功能	备注
PIT_CTRL	0x0043	PIT端口号	可编程间隔定时器
PIT_CNT0	0x0040	PIT端口号	
TIMER_MAX	500	定时器最大数量	
TIMER_FLAG_UNUSED	0	定时器未使用	
TIMER_FLAG_USED	1	定时器已使用	
TIMER_FLAG_USING	2	定时器正在使用	（启用，就绪）
TIMER_TIMER_MAX	42949673	定时器溢出上限	（十进制）
TIMER_TIME_MAX_H	0xffffffff	定时器溢出上限	（十六进制）

11.8 内存描述

常量名	值	功能	备注
MEMERY_ADDR	0x003c0000	内存首地址	
MEMERY_LIST_SIZE	409	内存空闲块表大小	（块）
SYS_PRE	0x00400000	系统预留内存	4MB
MEMERY_MAX_SIZE	0x07c00000	支持的最大内存容量	去除预留

11.9 内存处理

常量名	值	功能	备注
EFLAGS_AC_BITS	0x00040000	CPU类型校验码	486及以上CPU的EFLAGS寄存器有AC位
CR0_CACHE_DISABLE	0x60000000	禁用Cache的操作码	

11.10 文件信息

常量名	值	功能	备注
DISK_ADR	0x00100000	磁盘首地址	
FILE_INFO_MAX	224	文件信息最大个数	受FAT12文件系统限制
FILE_NAME_SIZE	8	文件名长度	
FILE_EXTEN_SIZE	3	扩展名长度	
FILE_FULL_NAME_SIZE	FILE_NAME_SIZE+FILE_EXTEN_SIZE	全名长度	

11.11 鼠标常量

常量名	值	功能	备注
KEYCMD_MOUSE	0xd4	键盘控制电路切换鼠标模式指令	
MOUSECMD_ENABLE	0xf4	鼠标激活指令	
ACK	0xfa	鼠标激活成功回答确认信息	
MOUSE_DATA_BASE	512	鼠标中断信号偏移	

11.12 键盘常量

常量名	值	功能	备注
PORT_KEYDAT	0x0060	键盘设备端口号	
PORT_KEYSTA	0x0064	键盘状态端口号	
PORT_KEYCMD	0x0064	键盘控制电路端口号	
KEYSTA_NOTREADY	0x02	键盘未准备好代码	准备好后，在键盘0x0064处读取的数据，从低位开始数第二位，即倒数第二位须为0
KEYCMD_WRITE_MODE	0x60	键盘控制电路模式设置指令	
KBC_MODE_MOUSE	0x47	键盘控制电路模式之鼠标模式	
KEY_DATA_BASE	256	键盘中断信号偏移	
KEYCMD_LED_STATUS	0xed	键盘初始化操作命令	硬件规定
KEY_TABLE_SIZE	0x80	键盘字符映射表大小	

11.13 缓冲区常量

常量名	值	功能	备注
FLAGS_OVERRUN	0x0001	溢出	
KEY_BUFFER_SIZE	32	键盘缓冲区大小	
MOUSE_BUFFER_SIZE	128	鼠标缓冲区大小	
TIMER_BUFFER_SIZE	8	定时器缓冲区大小	
BUFFER_SIZE	128	通用缓冲区大小	

11.14 桌面常量

常量名	值	功能	备注
DESK_BCOLOR	COL8_004276	桌面默认背景色	

11.15 图层常量

常量名	值	功能	备注
COVER_LIST_SIZE	256	图层表最大图层数	
COVER_UNUSE	0	图层未使用	
COVER_USEED	1	图层已使用	
COVER_NUM	15	图层最大图形数	

11.16 任务常量

常量名	值	功能	备注
TASK_MAX	1000	最大任务数	
TASK_GDT0	3	TSS的GDT起始号码	
TASK_FLAG_UNUSED	0	任务项未使用	
TASK_FLAG_USED	1	任务项已使用	（启用，就绪）
TASK_FLAG_RUNNING	2	任务正在运行	
TASK_LEVEL_MAX	100	任务优先级队列最大任务数	
LEVEL_MAX	10	优先级队列最大数量	

11.17 系统默认窗口定义

常量名	值	功能	备注
WINDOW_X	5	默认窗口横坐标	
WINDOW_Y	5	默认窗口纵坐标	
WINDOW_WIDTH	70	默认窗口宽度	
WINDOW_HEIGHT	50	默认窗口高度	
WINDOW_LINE_WIDTH	1	默认窗口边线宽度	
WINDOW_CAPTION_HEIGHT	20	默认窗口标题高度	
WINDOW_LINE_COLOR	COL8_101010	默认窗口边线颜色	（获得焦点时）
CONTROL_WINDOW	1	控制台窗口类型号	
NOMAL_WINDOW	2	通用窗口类型号	
INFO_WINDOW	3	信息提示窗口类型号	
TEXT_WINDOW	4	文本输入窗口类型号	
CONSOLE_WIDTH	400	默认控制台宽度	
CONSOLE_HEIGHT	250	默认控制台高度	
CONSOLE_FORECOLOR	COL8_38CE2F	默认控制台前景色	
REFRESH_ALL	0	窗口区域全部刷新操作数	
REFRESH_CAPTION	1	窗口标题局部刷新操作数	
CMD_CURSOR_X	16	默认控制台光标初始横坐标	
CMD_CURSOR_Y	28	默认控制台光标初始纵坐标	

11.18 系统颜色信息表

常量名	值	功能	备注
COL8_000000	0	纯黑	
COL8_FF0000	1	纯红	
COL8_00FF00	2	纯绿	
COL8_0000FF	3	纯蓝	
COL8_FFFF00	4	纯黄	
COL8_FF00FF	5	纯紫	
COL8_00FFFF	6	纯青	
COL8_FFFFFF	7	纯白	
COL8_C6C6C6	8	纯灰	
COL8_840000	9	暗红	
COL8_008400	10	暗绿	
COL8_000084	11	靛青	
COL8_848400	12	暗黄	
COL8_840084	13	暗紫	
COL8_008484	14	靛蓝	
COL8_848484	15	暗灰	
COL8_005B9E	16	湛蓝	
COL8_0078D7	17	浅蓝	
COL8_004276	18	深蓝	
COL8_FFFFFE	19	墨白	
COL8_E1E1E1	20	浅灰	
COL8_101010	21	明黑	
COL8_333333	22	黑灰	
COL8_D9D9D9	23	银灰	
COL8_E81123	24	亮红	
COL8_F0F0F0	25	薄灰	
COL8_38CE2F	26	亮青	

11.19 常用颜色别名

常量名	值	功能	备注
FRESH_BLUE	16	湛蓝	
TEEN_BLUE	17	浅蓝	
DEEP_BLUE	18	深蓝	
FLOUR_WHITE	19	墨白	
REAL_RED	1	纯红	
REAL_YELLOW	4	纯黄	
REAL_BLUE	3	纯蓝	
REAL_GREEN	2	纯绿	

11.20 字体常量

常量名	值	功能	备注
FONT_SIZE	16	字符位数常量	
FONT_CHAR	1	字符字节数常量	

11.21 其他公用常量

常量名	值	功能	备注
ZERO_ADR	0x00000000	特殊地址_零地址	
FULL_ADR	0xffffffff	特殊地址_最大地址	
TRUE	1		
FALSE	0		
NONE	0		

11.22 第三方临时常量

常量名	值	功能	备注
MEMMAN_FREES	4090	最大内存空闲块数（第三方）	
MEMMAN_ADDR	0x003c0000	内存首地址（第三方）	
MAX_SHEETS	256	最大图层数（第三方）	
APP_FLAG	"Hari"	Hari可执行文件标识	

11.23 专用类型定义

```
/*专用类型定义*/
typedef int COUNT;                    //定义COUNT类型（实质为int的别名，计数时使用，使之易于理解）
```

11.24 系统颜色信息数组

```
/*系统颜色信息*/ //使用静态一维数组存储数量有限的常用颜色，使用此函数绘图速度快
static unsigned char table_rgb[COLORNUM * 3] = { //静态无符号的16×3的颜色表格二维数组（节省空间，亦可加快绘制速度）。static char 只能用于数据
    0x00,0x00,0x00, //纯黑  0                    //颜色值定义为无符号数，防止意外溢出后由亮色变为暗色
    0xff,0x00,0x00, //纯红  1
    0x00,0xff,0x00, //纯绿  2
    0x00,0x00,0xff, //纯蓝  3
    0xff,0xff,0x00, //纯黄  4
    0xff,0x00,0xff, //纯紫  5
    0x00,0xff,0xff, //纯青  6
    0xff,0xff,0xff, //纯白  7
    0xc6,0xc6,0xc6, //纯灰  8
    0x84,0x00,0x00, //暗红  9
    0x00,0x84,0x00, //暗绿  10
    0x00,0x00,0x84, //靛青  11
    0x84,0x84,0x00, //暗黄  12
    0x84,0x00,0x84, //暗紫  13
    0x00,0x84,0x84, //靛蓝  14
    0x84,0x84,0x84, //暗灰  15
    0x00,0x5b,0x9e, //湛蓝  16
    0x00,0x78,0xd7, //浅蓝  17
    0x00,0x42,0x76, //深蓝  18
    0xff,0xff,0xfe, //墨白  19
    0xe1,0xe1,0xe1, //浅灰    20
    0x10,0x10,0x10, //明黑  21
    0x33,0x33,0x33, //黑灰  22
    0xd9,0xd9,0xd9, //银灰  23
    0xe8,0x11,0x23, //亮红    24
    0xf0,0xf0,0xf0, //薄灰  25
    0x38,0xce,0x2f, //亮青    26
    //浅黑 4c4a48
};
```

11.25 鼠标指针点阵

```
/*鼠标指针点阵*/
static char CUSOR_GRAPH[CURSOR_WIDTH][CURSOR_HEIGHT] = { //使用16*16点阵画出鼠标指针，点阵数据存在二维数组中
    "**00000000000000",
    "*1*0000000000000",
    "*111*00000000000",
    "*11111*000000000",
    "*1111111*0000000",
    "*111111111*00000",
    "*111111111110000",
    "*111111111111**0",
```

```
    "*1111111111111*",
    "*11111111*******",
    "*111111111*00000",
    "*1111*11111*0000",
    "*111*0*11111*000",
    "*11*000*1111*000",
    "*1*00000*111*000",
    "**0000000****000"
};
```

11.26 键盘映射数组

```
static char KEY_TABLE[KEY_TABLE_SIZE] = {
    0,0,'1','2','3','4','5','6','7','8','9','0','-','=',0,0,
    'Q','W','E','R','T','Y','U','I','O','P','[',']',0,0,
    'A','S','D','F','G','H','J','K','L',';','\'','`',0,'/',
    'Z','X','C','V','B','N','M',',','.','/',0,'*',0,' ',
    0,0,0,0,0,0,0,0,0,0,0,0,
    '7','8','9','-','4','5','6','+','1','2','3','0','.',
    0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0x5c,0, 0,0,0,0,0,0,0,0,0x5c,0,0
};

static char KEY_TABLE_CTRL[KEY_TABLE_SIZE] = {
    0,0,'!','@','#','$','%','^','&','*','(',')','_','+',0,0,
    'Q','W','E','R','T','Y','U','I','O','P','{','}',0,0,
    'A','S','D','F','G','H','J','K','L',':','\"','`',0,'/',
    'Z','X','C','V','B','N','M','<','>','?',0,'*',0,' ',
    0,0,0,0,0,0,0,0,0,0,0,0,
    '7','8','9','-','4','5','6','+','1','2','3','0','.',
    0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,'_',0, 0,0,0,0,0,0,0,0,'|',0,0
};
```

十二、其他参数

12.1 系统源文件列表及其主要作用

编号	文件名	层次	模块	功能简介	备注
01	ipl.asm	引导层	IPL	引导程序，引导系统启动。.nas同。	汇编语言
02	syshead.asm	引导层	SYSHEAD	系统实模式程序	汇编语言
03	osfun.asm	引导层	OSFUN	底层函数库，C下无法完成的实现	汇编语言
04	nnos.h	基础层	Manager Interface	头文件，系统函数声明	C语言
05	syshead.h	基础层	Manager Interface	系统结构体定义	C语言
06	systructual.h	基础层	Manager Interface	系统常量定义、静态数组定义	C语言
07	bootpack.c	核心层	BOOT	主程序，保护模式下运行	C语言
08	gdtidt.c	基础层	GDT&IDT	GDT和IDT相关函数实现	C语言
09	interrupt.c	基础层	INTERRUPT	中断相关函数实现	C语言
10	timer.c	基础层	TIMER	定时器相关函数实现	C语言
11	fifo.c	基础层	FIFO	先进先出缓冲区相关函数实现	C语言
12	memery.c	核心层	Memery Manager	内存管理函数实现	C语言
13	devices.c	核心层	I/O Manager	设备管理函数实现	C语言
14	task.c	核心层	Process Manger	进程管理和多任务函数实现	C语言
15	file.c	核心层	File Manger	文件管理函数实现	C语言
16	graphics.c	核心层	Graphics	图形绘制引擎、图形界面函数实现	C语言
17	fontbase.frc	扩展层	Font Base	基本点阵字体文件	字体文件
18	cover.c	扩展层	Layer Drawer	图层处理引擎	C语言

编号	文件名	层次	模块	功能简介	备注
19	console.c	应用层	Console	系统控制台应用程序	C语言
20	hello.nex	应用层	Other App	汇编语言应用程序示例	应用
21	hello02.nex	应用层	Other App	C语言应用程序示例	应用
22	hello03.nex	应用层	Other App	C语言调用API应用程序示例	应用
23	update.log	/	/	系统升级日志	日志
24	buglist.log	/	/	系统内核漏洞列表	日志
25	getnas.c	/	/	nasm兼容nask（Windows版）	C语言
26	getnas_linux.c	/	/	nasm兼容nask转换器（Linux版）	C语言
27	Makefile	/	/	文件生成规则，编译自动化	编译用脚本

12.2 寄存器及其用途规定

12.2.1 通用寄存器

汇编助记符	惯用名称		用途（规定）	备注
AX	Accumulator(累加寄存器)		通用，作累加器	可拆为AH和AL
DX	Data（数据寄存器）		通用，常存数据、	可拆为DH和DL
CX	Countor（计数寄存器）		通用，作计数器	可拆为CH和CL
BX	Base（基址寄存器）		通用，偏移	可拆，常与DS连用
SI	Source Index（源变址寄存器）		通用，源地址	串数据访问
DI	Destination Index		通用，目的地址	串数据访问
SP	Stack pointer（栈指针寄存器）		存放栈顶地址	用于访问堆栈数据
BP	Base pointer（基址指针寄存器）		存放栈帧地址	用于访问堆栈数据

> [注]在32位寄存器中与16位寄存器共用低16位地址，相应助记符前加“E”。表中名均略去“Register”。

12.2.2 段寄存器

汇编助记符	惯用名称	用途（规定）	备注
ES	Extra Segment（附加段寄存器）	存放附加段段基地址	段寄存器
DS	Data Segment（数据段寄存器）	存放数据段基地址	段寄存器
CS	Code Segment（代码段寄存器）	存放代码段基地址	段寄存器
SS	Stack Segment（栈寄存器）	存放堆栈段基地址	段寄存器
IP	Instruction Pointer（指令指针寄存器）	存放前须取的指令	专用，不可直接操作
FLAGS	状态标志寄存器。	存放处理器状态信息	专用寄存器

> [注]关于寄存器的详细介绍，请参看任意Intelx86处理器软件开发手册。

12.3 系统键盘字符映射表

序号	值	字符	ASCII	序号	值	字符	ASCII	序号	值	字符	ASCII		序号	值	字符	ASCII
0		空	00	27	]	]	1B	54		右Shift	36		81	3	小3	51
1		ESC	01	28		Enter	1C	55	*	小*	37		82	0	小0	52
2	1	1	02	29		左Ctrl	1D	56		左Alt	38		83	.	小.	53
3	2	2	03	30	A	A	1E	57		Space	39					
4	3	3	04	31	S	S	1F	58		CapsLock	3A					
5	4	4	05	32	D	D	20	59		F1	3B					
6	5	5	06	33	F	F	21	60		F2	3C					
7	6	6	07	34	G	G	22	61		F3	3D					
8	7	7	08	35	H	H	23	62		F4	3E					
9	8	8	09	36	J	J	24	63		F5	3F					
10	9	9	0A	37	K	K	25	64		F6	40					
11	0	0	0B	38	L	L	26	65		F7	41					
12	-	-	0C	39	;	;	27	66		F8	42					
13	=	=	0D	40	‘	‘	28	67		F9	43					
14		退格	0E	41	`	``	29	68		F10	44					
15		TAB	0F	42		左Shift	2A	69		Numlock	45					
16	Q	Q	10	43	\	\	2B	70		ScrollLock	46					
17	W	W	11	44	Z	Z	2C	71	7	小7	47					
18	E	E	12	45	X	X	2D	72	8	小8	48					
19	R	R	13	46	C	C	2E	73	9	小9	49					
20	T	T	14	47	V	V	2F	74	-	小-	4A					
21	Y	Y	15	48	B	B	30	75	4	小4	4B					
22	U	U	16	49	N	N	31	76	5	小5	4C					
23	I	I	17	50	M	M	32	77	6	小6	4D					
24	O	O	18	51	,	,	33	78	+	小+	4E					
25	P	P	19	52	.	.	34	79	1	小1	4F					
26	[	[	1A	53	/	/	35	80	2	小2	50					