**KalinoteOS 系统支持文档**

目录

[1. 说明 1](#_Toc60835640)

[2. 系统kernel工作方式 1](#_Toc60835641)

[2.1 系统内存管理方式 1](#_Toc60835642)

[2.2 FIFO数据缓冲区 1](#_Toc60835643)

[2.3 系统进程管理方式 2](#_Toc60835644)

[3. 系统内置CMD指令 2](#_Toc60835645)

[3.1 查询类 2](#_Toc60835646)

[mem指令 2](#_Toc60835647)

[dir&ls指令 2](#_Toc60835648)

[3.2 功能类 3](#_Toc60835649)

[cls&clear指令 3](#_Toc60835650)

[type [Filename]指令 3](#_Toc60835651)

[exit 指令 3](#_Toc60835652)

[start指令 3](#_Toc60835653)

[run指令 3](#_Toc60835654)

[langmode指令 3](#_Toc60835655)

[shutdown指令 3](#_Toc60835656)

[sysmode指令 3](#_Toc60835657)

[echo指令 4](#_Toc60835658)

[4. KalinoteOS系统API 4](#_Toc60835659)

[\_api\_putchar: 4](#_Toc60835660)

[\_api\_putstr0: 4](#_Toc60835661)

[cons\_putstr1: 4](#_Toc60835662)

[api\_end: 4](#_Toc60835663)

[api\_openwin: 5](#_Toc60835664)

[\_api\_putstrwin: 5](#_Toc60835665)

[\_api\_boxfilwin: 5](#_Toc60835666)

[\_api\_initmalloc: 6](#_Toc60835667)

[\_api\_malloc: 6](#_Toc60835668)

[\_api\_free: 6](#_Toc60835669)

[\_api\_point: 7](#_Toc60835670)

[\_api\_refreshwin: 7](#_Toc60835671)

[\_api\_linewin: 7](#_Toc60835672)

[\_api\_closewin: 8](#_Toc60835673)

[\_api\_getkey: 8](#_Toc60835674)

[\_api\_alloctimer: 8](#_Toc60835675)

[\_api\_inittimer: 8](#_Toc60835676)

[\_api\_settimer: 9](#_Toc60835677)

[\_api\_freetimer: 9](#_Toc60835678)

[\_api\_beep: 9](#_Toc60835679)

[\_api\_fopen: 9](#_Toc60835680)

[\_api\_fclose: 10](#_Toc60835681)

[\_api\_fseek: 10](#_Toc60835682)

[\_api\_fsize: 10](#_Toc60835683)

[\_api\_fread: 11](#_Toc60835684)

[\_api\_cmdline: 11](#_Toc60835685)

[\_api\_getlang: 11](#_Toc60835686)

[\_api\_cls: 11](#_Toc60835687)

[5. 标准函数API 12](#_Toc60835688)

[string.h 12](#_Toc60835689)

[malloc.h(stdlib.h) 12](#_Toc60835690)

[stdio.h 12](#_Toc60835691)

[ctype.h 12](#_Toc60835692)

[stdlib.h 12](#_Toc60835693)

[6. 后面要做的事 13](#_Toc60835694)

[压缩算法 13](#_Toc60835695)

[API函数标准化(标准函数) 13](#_Toc60835696)

# 说明

本文档是KalinoteOS的系统支持文档，在本文档中提到的KalinoteOS、系统、本系统、该系统等皆指KalinoteOS，在本文档中提到的程序、系统程序、软件等皆指在KalinoteOS上运行的kal或其他支持在KalinoteOS上运行的任何软件。

# 系统kernel工作方式

## 系统内存管理方式

KalinoteOS使用分段式内存管理方式，使用列表管理方法，原理是将可用内存信息以一个表的方式储存，储存的内容大概是类似”在XX号地址开始有XX字节空间可用”。为了应对不连续空间，系统中使用了一个MEMMAN\_FREES常量(该常量在bootpack.h中的memory.c部分)来建立多个表(系统设定了4090个表，约占32KB空间)用于储存可用内存段信息，每个表都可以保存一段空余内存信息，所以理论最多支持4090个内存碎片段。

例如：

Free[0]:地址0x00400000开始，有0x0A0F0510字节可用

Free[1]:地址0xAC105140开始，有0x0000D100字节可用

…

在内存碎片段占满MEMMAN\_FREES后，系统会先舍弃其他碎片空间，等到有碎片段释放后，重新进行内存检查，然后重新标记空内存段。

比如，某个程序需要分配512KB内存空间，则先遍历可用内存表，然后找到可用空间大于512KB的内存表，并将指向地址分配给程序。然后将可用地址表的指向地址更改为分配后的末尾，可用空间减少512KB。如果可用空间为0，则删除此表，MEMMAN->frees-1。

同样，在释放内存时，增加一条可用内存表，MEMMAN->frees +1。在释放内存时，还有可能会遇到两段内存表相邻(中间无不可用内存)，如：

Free[0]:地址0x00400000开始，有0x00019000字节可用

Free[1]:地址0x00419000开始，有0x07BE7000字节可用

…

以上两段内存可归纳为:

Free[0]:地址0x00400000开始，有0x07C00000字节可用

如果不将如上情况的两个表合并，则会不必要地消耗可用内存表，还有就是有程序在寻找可用内存时，有足够的可用空间却无法正常找到。

其中，可用内存表以结构体FREEINFO保存。

## FIFO数据缓冲区

在KalinoteOS中，会有多处设备(或程序)涉及到数据交换，每个设备进行数据交换的时间和数值都会有所不同。在本系统中有几种数据管理方式，其中一种是队列(FIFO,First in first out)数据管理方式，在fifo结构体中有以下几个变量：

int \*buf，一个地址指针，用来指向缓冲区地址

int p，表示下一次数据的写入地址

int q，表示下一次数据的读入地址

int size，表示缓冲区大小

int free，表示缓冲区空闲字节数

int flags，表示溢出标志

struct TASK \*task 有数据写入时需要唤醒的任务，TASK是任务结构体，详细可以参考后面的系统进程管理方式一节

在系统主进程(task\_a)中，有一个专用的fifo缓冲区，用于处理各项事务(bootpack.c中的struct FIFO32 fifo)，系统中包括鼠标、键盘、时间、命令行相关操作等需要在系统主循环中处理的数据皆使用这个缓冲区。对于这个缓冲区，不同控制会使用不同的控制区间，详细如下：

1 系统时间

256-511 键盘数据(键盘控制器读入值+256)

512-767 鼠标数据(鼠标控制器读入值+512)

768-1023 命令行：窗口关闭处理

1024-2023 命令行：结束任务

2024-2279 命令行：关闭命令窗口，不结束任务

## 系统进程管理方式

# 系统内置CMD指令

## 查询类

### mem指令

该指令为memory的缩写，作用是查询系统内存使用情况。

### dir&ls指令

该指令用于查询系统中的文件。

## 功能类

### cls&clear指令

cls为clear screen的缩写，该指令的作用是清空命令行窗口。

### type [Filename]指令

type指令用于输出某个文件的内容。

### exit 指令

该指令用于关闭命令窗口。

### start指令

该指令用于在新的命令窗口启动一个应用程序。

### run指令

该指令用于在当前命令窗口执行一个应用程序，并不占用当前命令窗口。

### langmode指令

该指令用于切换系统显示语言模式。

### shutdown指令

该指令用于关闭计算机。

### sysmode指令

该指令用于切换系统模式。.

### echo指令

该指令用于系统输出。

# KalinoteOS系统API

### \_api\_putchar:

寄存器数据：

EDX = 1

AL = 字符

描述：

在命令窗口打印单个字符。

调用方法：

**void api\_putchar(int c);**

### \_api\_putstr0:

寄存器数据：

EDX = 2

EBX = 字符串

描述：

在命令窗口打印字符串，并在结尾为字符编码0时结束。

调用方法：

**void api\_putstr0(char \*s);**

### cons\_putstr1:

寄存器数据：

EDX = 3

EBX = 字符串

描述：

在命令窗口打印字符串，并提前指定字符串长度。

调用方法：

停止使用

### api\_end:

寄存器数据：

EDX = 4

描述：

用于结束运行应用程序。

调用方法：

**void api\_end(void);**

### api\_openwin:

寄存器数据：

EDX = 5

EBX = 窗口缓冲区

ESI = 窗口X(高度)

EDI = 窗口Y(宽度)

EAX = 透明色

ECX = 窗口名称

描述：

用于生成一个基本窗口。

调用方法：

**int api\_openwin(char \*buf, int xsiz, int ysiz, int col\_inv, char \*title);**

### \_api\_putstrwin:

寄存器数据：

EDX = 6

EBX = 窗口句柄

ESI = 显示字符串X坐标

EDI = 显示字符串Y坐标

EAX = 色号

ECX = 字符串长度

EBP = 字符串

描述：

用于在指定窗口上打印一个字符串。

调用方法：

**void api\_putstrwin(int win, int x, int y, int col, int len, char \*str);**

### \_api\_boxfilwin:

寄存器数据：

EDX = 7

EBX = 窗口句柄

EAX =

ECX =

ESI =

EDI =

EBP = 色号

描述：

用于在指定窗口上打印一个字符串。

调用方法：

**void api\_boxfilwin(int win, int x0, int y0, int x1, int y1, int col);**

### \_api\_initmalloc:

寄存器数据：

EDX = 8

EBX = memman地址

EAX = memman所管理的内存空间的起始地址

ECX = memman所管理的内存空间的字节数

描述：

用于初始化内存分配程序。

调用方法：

**void api\_initmalloc(void);**

### \_api\_malloc:

寄存器数据：

EDX = 9

EBX = memman的地址

ECX = 需要请求的字节数

EAX = 分配到的内存空间地址

描述：

用于分配内存空间。

调用方法：

**char \*api\_malloc(int size);**

### \_api\_free:

寄存器数据：

EDX = 10

EBX = memman地址

EAX = 需要释放的内存空间地址

ECX = 需要释放的字节数

描述：

用于释放内存空间。

调用方法：

**void api\_end(void);**

### \_api\_point:

寄存器数据：

EDX = 11

EBX = 窗口句柄

ESI = 显示位置的X坐标

EDI = 显示位置的Y坐标

EAX = 色号

描述：

在指定窗口的某个位置描绘一个像素点。

调用方法：

**void api\_point(int win, int x, int y, int col);**

### \_api\_refreshwin:

寄存器数据：

EDX = 12

EBX = 窗口句柄

EAX =

ECX =

ESI =

EDI =

描述：

用于窗口的区域渲染，在窗口绘制完成后刷新以显示。

使用方法：

**void api\_refreshwin(int win, int x0, int y0, int x1, int y1);**

### \_api\_linewin:

寄存器数据：

EDX = 13

EBX = 窗口句柄

EAX =

ECX =

ESI =

EDI =

EBP = 色号

描述：

用于窗口直线渲染。

使用方法：

**void api\_linewin(int win, int x0, int y0, int x1, int y1, int col);**

### \_api\_closewin:

寄存器数据：

EDX = 14

EBX = 窗口句柄

描述：

用于关闭窗口。

使用方法：

**void api\_closewin(int win);**

### \_api\_getkey:

寄存器数据：

EDX = 15

EAX = 获取模式

EAX = 键盘输入的字符编码

描述：

用于获取键盘数据，关于获取模式：参数为0的话该函数在没有键盘输入时返回 -1，参数为1的话该函数在没有键盘输入时休眠。

使用方法：

**int api\_getkey(int mode);**

### \_api\_alloctimer:

寄存器数据：

EDX = 16

EAX = 定时器句柄(由系统返回)

描述：

从系统申请一个定时器。

使用方法：

**int api\_alloctimer(void);**

### \_api\_inittimer:

寄存器数据：

EDX = 17

EBX = 定时器句柄

EAX = 数据

描述：

设定一个数据，在定时器超时后会返回这个数据。

使用方法：

**void api\_inittimer(int timer, int data);**

### \_api\_settimer:

寄存器数据：

EDX = 18

EBX = 定时器句柄

EAX = 时间(ms，毫秒)

描述：

设定一个时间，产生延时。

使用方法：

**void api\_settimer(int timer, int time);**

### \_api\_freetimer:

寄存器数据：

EDX = 19

EBX = 定时器句柄

描述：

释放定时器。

使用方法：

**void api\_freetimer(int timer);**

### \_api\_beep:

寄存器数据：

EDX = 20

EAX = 声音频率(0为停止)

描述：

控制蜂鸣器发声。

使用方法：

**void api\_beep(int tone);**

### \_api\_fopen:

寄存器数据：

EDX = 21

EBX = 文件名

EAX = 文件句柄(为0时表示打开失败，由操作系统返回)

描述：

用于打开一个文件。

使用方法：

**int api\_fopen(char \*fname);**

### \_api\_fclose:

寄存器数据：

EDX = 22

EAX = 文件句柄

描述：

用于关闭一个文件。

使用方法：

**void api\_fclose(int fhandle);**

### \_api\_fseek:

寄存器数据：

EDX = 23

EAX = 文件句柄

ECX = 定位模式

EBX = 定位偏移量

描述：

用于寻找文件时的文件定位，其中定位模式有以下几种：

ECX = 0 定位的起点为文件开头

ECX = 1 定位的起点为当前访问的位置

ECX = 2 定位的起点为文件末尾

使用方法：

**void api\_fseek(int fhandle, int offset, int mode);**

### \_api\_fsize:

寄存器数据：

EDX = 24

EAX = 文件句柄

ECX = 文件大小获取模式

描述：

用于获取文件大小，其中获取模式有以下几种：

ECX = 0 普通文件大小

ECX = 1 当前读取位置从文件开头起算的偏移量

ECX = 2 当前读取位置从文件末尾起算的偏移量

使用方法：

**int api\_fsize(int fhandle, int mode);**

### \_api\_fread:

寄存器数据：

EDX = 25

EAX = 文件句柄

EBX = 缓冲区地址

ECX = 最大读取字节数

EAX = 本次读取到的字节数(由系统返回)

描述：

用于读取文件。

使用方法：

**int api\_fread(char \*buf, int maxsize, int fhandle);**

### \_api\_cmdline:

寄存器数据：

EDX = 26

EBX = 存放命令行内容的地址

ECX = 最多可以放多少字节

EAX = 实际存放了多少字节(操作系统返回)

描述：

用于获取用户在命令行输入的参数

使用方法：

**int api\_cmdline(char \*buf, int maxsize);**

### \_api\_getlang:

寄存器数据：

EDX = 27

EAX = langmode(由系统返回)

描述：

查询当前系统语言模式。

使用方法：

**int api\_getlang(void);**

### \_api\_cls:

寄存器数据：

EDX = 29

描述：

清空cmd内容（这个api暂时不能使用）。

使用方法：

void api\_cls(void);

# 标准函数API

下面是KalinoteOS支持的C语言标准API，具体使用方法可以查询C语言支持手册。

### string.h

int strchr(const char \*str, char c);

### malloc.h(stdlib.h)

void \*malloc(int size);

void free(void \*p);

### stdio.h

int putchar(int c);

int printf(char \*format, ...);

int scanf(const char \*format, ...);

int getchar();

int puts(const char \*str);

char \*gets(char \*str);

### ctype.h

int isspace(char c);

int isdigit(char c);

### stdlib.h

void exit(int status);

# 后面要做的事

### 压缩算法

### API函数标准化(标准函数)