库函数

ELF

定义

```
#ifndef _ELF_H
#define _ELF_H

#include <stdint.h>
/* Type for a 16-bit quantity.  */
typedef uint16_t Elf32_Half;

/* Types for signed and unsigned 32-bit quantities.  */
typedef uint32_t Elf32_Word;
typedef int32_t Elf32_Sword;

/* Types for signed and unsigned 64-bit quantities.  */
typedef uint64_t Elf32_Xword;
typedef int64_t Elf32_Sxword;

/* Type of addresses.  地址 */
typedef uint32_t Elf32_Addr;

/* Type of file offsets. 偏移 */
typedef uint32_t Elf32_Off;

/* Type for section indices, which are 16-bit quantities.  */
typedef uint16_t Elf32_Section;

/* Type of symbol indices.  */
typedef uint32_t Elf32_Symndx;

/* The ELF file header.  This appears at the start of every ELF file.  */

#define EI_NIDENT (16)

typedef struct {
    unsigned char e_ident[EI_NIDENT]; /* Magic number and other info */
    Elf32_Half e_type;        /* Object file type */
    Elf32_Half e_machine;         /* Architecture */
    Elf32_Word e_version;         /* Object file version */
    Elf32_Addr e_entry;       /* Entry point virtual address */
    Elf32_Off e_phoff;        /* Program header table file offset */
    Elf32_Off e_shoff;        /* Section header table file offset */
    Elf32_Word e_flags;       /* Processor-specific flags */
    Elf32_Half e_ehsize;          /* ELF header size in bytes */
    Elf32_Half e_phentsize;       /* Program header table entry size */
    Elf32_Half e_phnum;       /* Program header table entry count */
    Elf32_Half e_shentsize;       /* Section header table entry size */
    Elf32_Half e_shnum;       /* Section header table entry count */
    Elf32_Half e_shstrndx;        /* Section header string table index */
} Elf32_Ehdr; // ELF头
```

```c
/* Fields in the e_ident array.  The EI_* macros are indices into the
   array.  The macros under each EI_* macro are the values the byte
   may have.  */

#define EI_MAG0 0    /* File identification byte 0 index */
#define ELFMAG0 0x7f /* Magic number byte 0 */

#define EI_MAG1 1   /* File identification byte 1 index */
#define ELFMAG1 'E' /* Magic number byte 1 */

#define EI_MAG2 2   /* File identification byte 2 index */
#define ELFMAG2 'L' /* Magic number byte 2 */

#define EI_MAG3 3   /* File identification byte 3 index */
#define ELFMAG3 'F' /* Magic number byte 3 */

/* Section segment header.
主要用于链接
*/
typedef struct {
    Elf32_Word sh_name;  /* Section name 表示节名在字符串表中的索引 */
    Elf32_Word sh_type;  /* Section type 包含代码的可执行段、包含数据的数据段或者其他一些特殊类型的段，如符号表、字符串表等 */
    Elf32_Word sh_flags;     /* Section flags 例如，它可以表示该节是否可读、可写或可执行等 */
    Elf32_Addr sh_addr;  /* 表示该节在内存中的虚拟地址。当程序加载到内存时，这个值表示该节应该被映射到的内存地址 */
    Elf32_Off sh_offset;     /* 表示该节在ELF文件中的偏移量。这个值用于在文件中找到该节的数据 */
    Elf32_Word sh_size;  /* Section size 表示该节在内存中的大小（字节数） */
    Elf32_Word sh_link;  /* Section link 表示与该节相关联的另一个节的节头部索引，具体的含义取决于节的类型 */
    Elf32_Word sh_info;  /* Section extra info 表示与该节相关的额外信息，具体的含义取决于节的类型 */
    Elf32_Word sh_addralign; /* Section alignment */
    Elf32_Word sh_entsize;   /* Section entry size 表示该节中的表项大小。对于包含固定大小表项的节（如符号表），这个值表示每个表项的大小；如果节不包含表项，该值为0 */
} Elf32_Shdr; // 节头表表项

/* Program segment header.
程序段是ELF文件的另一个逻辑单位，主要用于描述可执行文件和共享库在加载到内存时的映射关系。
主要用于加载和执行
*/
typedef struct {
    Elf32_Word p_type;   /* Segment type 它可以是一个包含代码或数据的加载段（LOAD）、动态链接信息段（DYNAMIC）等 */
    Elf32_Off p_offset;  /* 表示程序段在ELF文件中的偏移量。这个值用于在文件中找到该程序段的数据 */
    Elf32_Addr p_vaddr;  /* 表示该程序段在内存中的虚拟地址。当程序加载到内存时，这个值表示该程序段应该被映射到的内存地址 */
    Elf32_Addr p_paddr;  /* 表示程序段在内存中的物理地址。对于大多数系统，这个字段的值并不重要，因为程序加载过程中使用的是虚拟地址 */
    Elf32_Word p_filesz; /* 表示程序段在文件中的大小（字节数）。这个值表示从文件中读取的数据量 */
    Elf32_Word p_memsz;  /* 表示程序段在内存中的大小（字节数） */
```

```
    Elf32_Word p_flags;  /* 可以表示该程序段是否可读、可写或可执行等 */
    Elf32_Word p_align;  /* 表示该程序段在内存和文件中m的对齐约束 */
} Elf32_Phdr; // 段头

/* Legal values for p_type (segment type).  */

#define PT_NULL 0        /* Program header table entry unused */
#define PT_LOAD 1        /* Loadable program segment */
#define PT_DYNAMIC 2         /* Dynamic linking information */
#define PT_INTERP 3      /* Program interpreter */
#define PT_NOTE 4        /* Auxiliary information */
#define PT_SHLIB 5       /* Reserved */
#define PT_PHDR 6        /* Entry for header table itself */
#define PT_NUM 7         /* Number of defined types.  */
#define PT_LOOS 0x60000000   /* Start of OS-specific */
#define PT_HIOS 0x6fffffff   /* End of OS-specific */
#define PT_LOPROC 0x70000000 /* Start of processor-specific */
#define PT_HIPROC 0x7fffffff /* End of processor-specific */

/* Legal values for p_flags (segment flags).  */

#define PF_X (1 << 0)          /* Segment is executable */
#define PF_W (1 << 1)          /* Segment is writable */
#define PF_R (1 << 2)          /* Segment is readable */
#define PF_MASKPROC 0xf0000000 /* Processor-specific */

#endif /* elf.h */
```

功能宏

```
// include/elf.h
// for循环迭代遍历所有段头
#define ELF_FOREACH_PHDR_OFF(ph_off, ehdr)
                  \
    (ph_off) = (ehdr)->e_phoff;
            \
    for (int _ph_idx = 0; _ph_idx < (ehdr)->e_phnum; ++_ph_idx, (ph_off) +=
(ehdr)->e_phentsize)
```

内核崩溃宏

```
// include/printk.h

//是一个内部函数，用于打印错误信息并停止系统运行。如果在 MOS_HANG_ON_PANIC 宏被定义的情况下，
系统会一直处于停滞状态；否则会退出程序
void _panic(const char *, int, const char *, const char *, ...)
#ifdef MOS_HANG_ON_PANIC
    __attribute__((noreturn))
#endif
    ;

//它将 __FILE__、__LINE__ 和 __func__ 作为参数传递给 _panic 函数，同时还接受可变参数列表。
当程序运行到这个宏时，如果表达式的值为非零，就会触发 _panic 函数停止系统运行
#define panic(...) _panic(__FILE__, __LINE__, __func__, __VA_ARGS__)
```

```
//用于在表达式的值为非零时调用 panic 宏并传递相应的错误信息。如果表达式的值为零，就不会执行任何操作。这个宏主要用于简化代码，避免在多处编写相似的 if 语句来检查错误。
#define panic_on(expr)                  \
    do {                                \
        int r = (expr);                 \
        if (r != 0) {                   \
            panic("'" #expr "' returned %d", r);  \
        }                               \
    } while (0)
```

printk

变长参数宏

```
va_list ap;  // 变长参数表
va_start(ap, lastarg);  // 使用变长参数表的前一个参数，初始化变长参数表

int num;
num = va_arg(ap, int);  // 获取下一个参数

va_end(ap);  //结束使用变长参数表
```