

工作笔记

段武杰

September 10, 2016

内存管理

1.1 虚拟地址转换

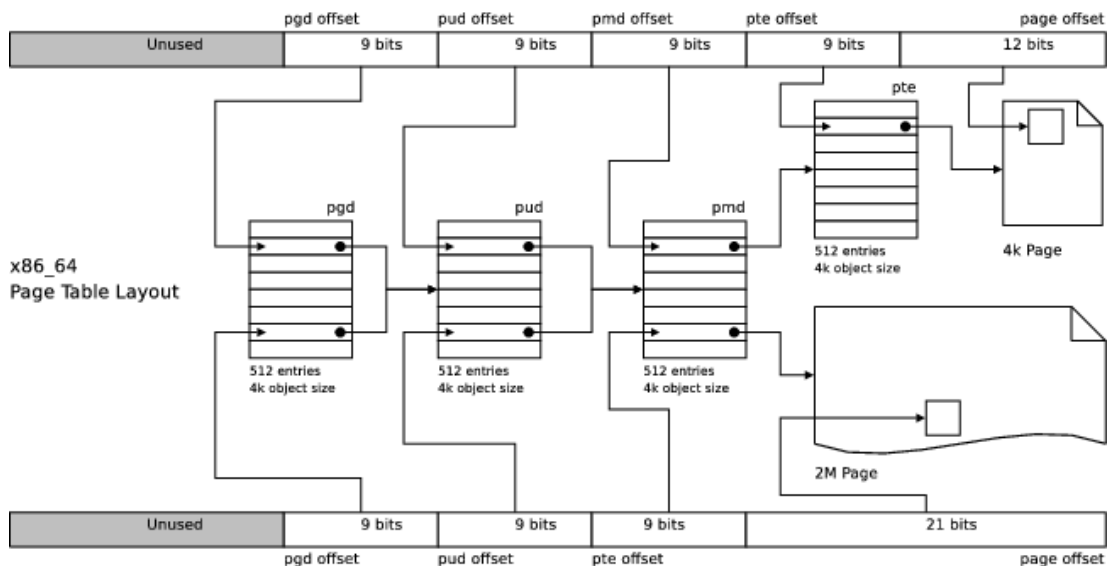


Figure 1.1: 虚拟地址转换 [1]

文件系统

目标：能够自己写一个文件系统。

VFS 使用面向对象的设计思路，VFS 中有 4 个主要的对象类型：

- 超级块对象 (super_block): 它表示一个具体的已安装的文件系统
- 索引节点对象 (inode): 它表示一个具体的文件
- 目录项对象 (dentry): 它表示一个目录项，是路径的一个组成部分。
- 文件对象 (file): 它表示进程打开的文件。

VFS 将目录当作文件来处理，所以不存在目录对象，目录项代表的是路径中的一个组成部分。

2.1 file_system_type

内核使用该结构体来描述文件系统的功能和行为：

```
struct file_system_type {  
    const char *name; /* 文件系统名字 */  
};
```

```

    int fs_flags; /* 文件系统类型标志 */
#define FS_REQUIRES_DEV      1
#define FS_BINARY_MOUNTDATA  2
#define FS_HAS_SUBTYPE       4
#define FS_USERSNS_MOUNT     8 /* Can be mounted by usersns root */
#define FS_USERSNS_DEV_MOUNT 16 /* A usersns mount does not imply MNT_NODEV */
#define FS_USERSNS_VISIBLE   32 /* FS must already be visible */
#define FS_RENAME_DOES_D_MOVE 32768 /* FS will handle d_move() during rename() internally. */
    struct dentry *(*mount) (struct file_system_type *, int,
                            const char *, void *);
    void (*kill_sb) (struct super_block *); /* 用来终止访问 super_block */
    struct module *owner; /* 文件系统模块 */
    struct file_system_type * next;
    struct hlist_head fs_supers; /* supperblock list */

    /* Runtime lock */
    struct lock_class_key s_lock_key;
    struct lock_class_key s_umount_key;
    struct lock_class_key s_vfs_rename_key;
    struct lock_class_key s_writers_key[SB_FREEZE_LEVELS];

    struct lock_class_key i_lock_key;
    struct lock_class_key i_mutex_key;
    struct lock_class_key i_mutex_dir_key;
};

```

2.2 inode

内核处理文件的关键是 inode，每个文件（和目录）都有且只有一个对应的 inode，其中包含元数据（如访问权限，上次修改的日期，等等）和指向文件数据的指针。

```

/*
 * Keep mostly read-only and often accessed (especially for
 * the RCU path lookup and 'stat' data) fields at the beginning
 * of the 'struct inode'
 */
struct inode { /* fs.h */
    umode_t          i_mode; /* 文件访问权限和所有权 */
    unsigned short    i_opflags;
    kuid_t           i_uid; /* uid about the file */
    kgid_t           i_gid; /* gid about the file */
    unsigned int      i_flags;

```

```

#ifdef CONFIG_FS_POSIX_ACL
    struct posix_acl      *i_acl;
    struct posix_acl      *i_default_acl;
#endif

/* 负责管理结构性操作（如删除一个文件）和文件相关的元数据例如属性() */
const struct inode_operations *i_op;
struct super_block      *i_sb;
struct address_space     *i_mapping;

#ifdef CONFIG_SECURITY
    void                  *i_security;
#endif

/* Stat data, not accessed from path walking */
/* 对给定的文件系统，唯一的编号标识 */
unsigned long            i_ino;
/*
 * Filesystems may only read i_nlink directly. They shall use the
 * following functions for modification:
 *
 * (set|clear|inc|drop)_nlink
 * inode_(inc|dec)_link_count
 */
union {
    /* 记录使用该 inode 的硬链接总数 */
    const unsigned int i_nlink;
    unsigned int __i_nlink;
};
dev_t                    i_rdev;

loff_t                   i_size; /* 文件大小 */
struct timespec          i_atime; /* 最后访问时间 */
struct timespec          i_mtime; /* 最后修改时间 */
struct timespec          i_ctime; /* inode 最后修改时间 */
spinlock_t               i_lock; /* i_blocks, i_bytes, maybe i_size */
unsigned short           i_bytes;
unsigned int              i_blkbits;
blkcnt_t                 i_blocks; /* 指定了按块存放的长度 */

#ifdef __NEED_I_SIZE_ORDERED
    seqcount_t            i_size_seqcount;
#endif

/* Misc */
unsigned long             i_state;
struct mutex              i_mutex;

unsigned long             dirtied_when; /* jiffies of first dirtying */
unsigned long             dirtied_time_when;

struct hlist_node         i_hash;
struct list_head          i_io_list; /* backing dev IO list */

```

```

#ifdef CONFIG_CGROUP_WRITEBACK
    struct bdi_writeback    *i_wb;          /* the associated cgroup wb */

    /* foreign inode detection, see wbc_detach_inode() */
    int                     i_wb_frn_winner;
    u16                     i_wb_frn_avg_time;
    u16                     i_wb_frn_history;
#endif

    struct list_head        i_lru;          /* inode LRU list */
    struct list_head        i_sb_list;
    union {
        struct hlist_head    i_dentry;
        struct rcu_head      i_rcu;
    };

    u64                     i_version;
    atomic_t                i_count; /* 访问该的进程数目inode */
    atomic_t                i_dio_count;
    atomic_t                i_writecount;
#ifdef CONFIG_IMA
    atomic_t                i_readcount; /* struct files open R0 */
#endif

    const struct file_operations *i_fop; /* 用于操作文件中包含的数据 */
    struct file_lock_context    *i_flctx;
    struct address_space        i_data;
    struct list_head            i_devices;
    union {
        struct pipe_inode_info *i_pipe;
        struct block_device    *i_bdev;
        struct cdev             *i_cdev;
        char                    *i_link;
    };

    __u32                    i_generation;

#ifdef CONFIG_FSNOTIFY
    __u32                    i_fsnotify_mask; /* all events this inode cares about */
    struct hlist_head        i_fsnotify_marks;
#endif

    void                     *i_private; /* fs or device private pointer */
};

```

2.3 inode_operations

大多数情况下，各个函数指针成员的意义可以根据其名称推断。它们与对应的系统调用和用户空间工具在名称方面非常相似。

```

struct inode_operations {
    /* lookup 根据文件系统对象的名称表示为字符串 ) 查找其( inode 实例*/
    struct dentry * (*lookup) (struct inode *,struct dentry *, unsigned int);
    const char * (*follow_link) (struct dentry *, void **);
    int (*permission) (struct inode *, int);
    struct posix_acl * (*get_acl)(struct inode *, int);

    int (*readlink) (struct dentry *, char __user *,int);
    void (*put_link) (struct inode *, void *);

    int (*create) (struct inode *,struct dentry *, umode_t, bool);
    int (*link) (struct dentry *,struct inode *,struct dentry *);
    int (*unlink) (struct inode *,struct dentry *);
    int (*symlink) (struct inode *,struct dentry *,const char *);
    int (*mkdir) (struct inode *,struct dentry *,umode_t);
    int (*rmdir) (struct inode *,struct dentry *);
    int (*mknod) (struct inode *,struct dentry *,umode_t,dev_t);
    int (*rename) (struct inode *, struct dentry *,
                    struct inode *, struct dentry *);
    int (*rename2) (struct inode *, struct dentry *,
                    struct inode *, struct dentry *, unsigned int);
    int (*setattr) (struct dentry *, struct iattr *);
    int (*getattr) (struct vfsmount *mnt, struct dentry *, struct kstat *);
    int (*setxattr) (struct dentry *, const char *,const void *,size_t,int);
    ssize_t (*getxattr) (struct dentry *, const char *, void *, size_t);
    ssize_t (*listxattr) (struct dentry *, char *, size_t);
    int (*removexattr) (struct dentry *, const char *);
    int (*fiemap)(struct inode *, struct fiemap_extent_info *, u64 start,
                  u64 len);

    int (*update_time)(struct inode *, struct timespec *, int);
    int (*atomic_open)(struct inode *, struct dentry *,
                       struct file *, unsigned open_flag,
                       umode_t create_mode, int *opened);
    int (*tmpfile) (struct inode *, struct dentry *, umode_t);
    int (*set_acl)(struct inode *, struct posix_acl *, int);

    /* WARNING: probably going away soon, do not use! */
} ____cacheline_aligned;

```

2.4 super_block

内核使用该结构体来描述文件系统的功能和行为：

```

struct super_block {
    struct list_head    s_list;        /* 指向的链表super_block */
    dev_t               s_dev;         /* 设备标识符 */

```

```

unsigned char      s_blocksize_bits; /* 以位为单位的块大小 */
unsigned long      s_blocksize; /* 以字节为单位的块大小 */
loff_t            s_maxbytes; /* Max file size */
struct file_system_type *s_type; /* Filesystem type */
const struct super_operations *s_op; /* 超级块方法 */
const struct dqquot_operations *dq_op; /* 磁盘限额方法 */
const struct quotactl_ops *s_qcop; /* 限额控制方法 */
const struct export_operations *s_export_op; /* 导出方法 */
unsigned long      s_flags; /* 挂载标志 */
unsigned long      s_iflags; /* internal SB_I_* flags */
unsigned long      s_magic; /* 文件系统魔数 */
struct dentry      *s_root; /* 目录挂载点 */
struct rw_semaphore s_umount; /* 卸载信号量 */
int               s_count; /* 超级块引用计数 */
atomic_t          s_active; /* 活动引用计数 */
#ifdef CONFIG_SECURITY
void              *s_security; /* 安全模块 */
#endif

const struct xattr_handler **s_xattr; /* 扩展的属性操作 */

struct hlist_bl_head s_anon; /* anonymous dentries for (nfs) exporting */
struct list_head     s_mounts; /* list of mounts; _not_ for fs use */
struct block_device *s_bdev; /* 相关的块设备 */
struct backing_dev_info *s_bdi;
struct mtd_info      *s_mtd;
struct hlist_node     s_instances;
unsigned int         s_quota_types; /* Bitmask of supported quota types */
struct quota_info     s_dquot; /* Diskquota specific options */

struct sb_writers     s_writers;

char s_id[32]; /* Informational name */
u8 s_uuid[16]; /* UUID */

void *s_fs_info; /* Filesystem private info */
unsigned int s_max_links;
fmode_t s_mode;

/* Granularity of c/m/atime in ns.
   Cannot be worse than a second */
u32 s_time_gran;

/*
 * The next field is for VFS *only*. No filesystems have any business
 * even looking at it. You had been warned.
 */
struct mutex s_vfs_rename_mutex; /* Kludge */

/*
 * Filesystem subtype. If non-empty the filesystem type field
 * in /proc/mounts will be "type.subtype"
 */

```



```

char *s_subtype;

/*
 * Saved mount options for lazy filesystems using
 * generic_show_options()
 */
char __rcu *s_options;
const struct dentry_operations *s_d_op; /* default d_op for dentries */

/*
 * Saved pool identifier for cleancache (-1 means none)
 */
int cleancache_poolid;

struct shrinker s_shrink;      /* per-sb shrinker handle */

/* Number of inodes with nlink == 0 but still referenced */
atomic_long_t s_remove_count;

/* Being remounted read-only */
int s_readonly_remount;

/* AIO completions deferred from interrupt context */
struct workqueue_struct *s_dio_done_wq;
struct hlist_head s_pins;

/*
 * Keep the lru lists last in the structure so they always sit on their
 * own individual cachelines.
 */
struct list_lru      s_dentry_lru ____cacheline_aligned_in_smp;
struct list_lru      s_inode_lru  ____cacheline_aligned_in_smp;
struct rcu_head       rcu;
struct work_struct    destroy_work;

struct mutex          s_sync_lock; /* sync serialisation lock */

/*
 * Indicates how deep in a filesystem stack this SB is
 */
int s_stack_depth;

/* s_inode_list_lock protects s_inodes */
spinlock_t            s_inode_list_lock ____cacheline_aligned_in_smp;
struct list_head       s_inodes;      /* all inodes */
};

```

2.5 dentry

```
struct dentry {
    /* RCU lookup touched fields */
    unsigned int d_flags;          /* protected by d_lock */
    seqcount_t d_seq;             /* per dentry seqlock */
    struct hlist_bl_node d_hash;   /* lookup hash list */
    struct dentry *d_parent;       /* parent directory */
    struct qstr d_name;
    struct inode *d_inode;         /* Where the name belongs to - NULL is
                                   * negative */
    unsigned char d_iname[DNAME_INLINE_LEN]; /* small names */

    /* Ref lookup also touches following */
    struct lockref d_lockref;      /* per-dentry lock and refcount */
    const struct dentry_operations *d_op;
    struct super_block *d_sb;      /* The root of the dentry tree */
    unsigned long d_time;          /* used by d_revalidate */
    void *d_fsdata;               /* fs-specific data */

    struct list_head d_lru;        /* LRU list */
    struct list_head d_child;      /* child of parent list */
    struct list_head d_subdirs;    /* our children */
    /*
     * d_alias and d_rcu can share memory
     */
    union {
        struct hlist_node d_alias; /* inode alias list */
        struct rcu_head d_rcu;
    } d_u;
};
```

中断处理

中断的本质是一种特殊的电信号，有硬件发向处理器。内核启用中断以前，必须把 IDT 表的初始化地址装到 `idt` 寄存器，并初始化表中的每一项。

模板

```
1 int main(int argc, char ** argv)
2 {
3     printf("Hello world!\n");
4     return 0;
5 }
```

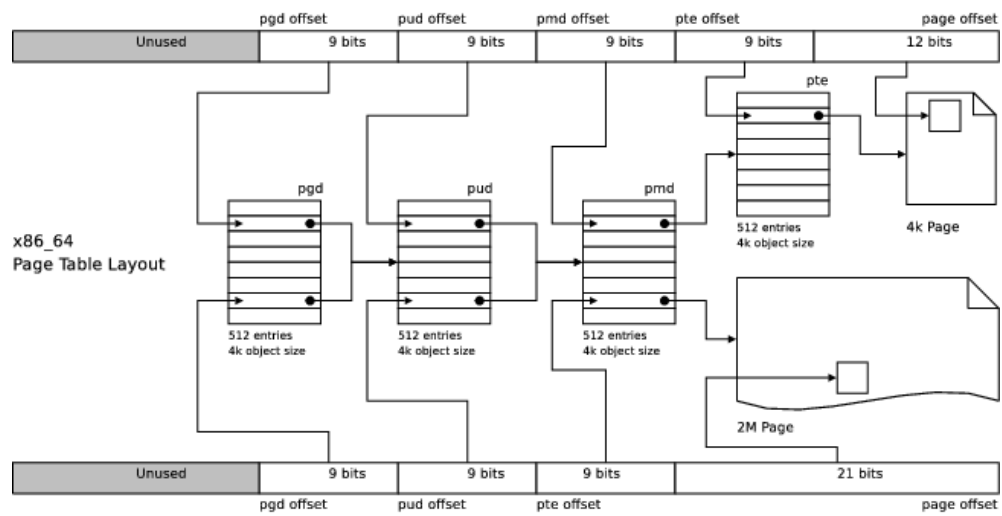


Figure 4.1: 虚拟地址转换

Bibliography

- [1] <http://linux-mm.org/PageTableStructure>. 1