# Chapter I Introduction

As the operating system is the foundation of all application software, learning system is not only critical to system software, but also to the improvement of application software level. File system plays an important role in the Linux kernel. Learning from it will help us improve program efficiency.

### Research background and significance

File system is the method and data structure used by the operating system to define the files on the storage device or partition; That is, the method of organizing files on storage devices. The software organization responsible for managing and storing file information in the operating system is called file management system, or file system for short. The file system consists of three parts: the interface of the file system, the software collection for object manipulation and management, and objects and attributes. From the perspective of system, file system is a system that organizes and allocates the space of file storage devices, is responsible for file storage, and protects and retrieves the stored files. Specifically, it is responsible for creating files for users, storing, reading, modifying, and dumping files, controlling file access, and revoking files when users no longer use them.

### The research content of this paper

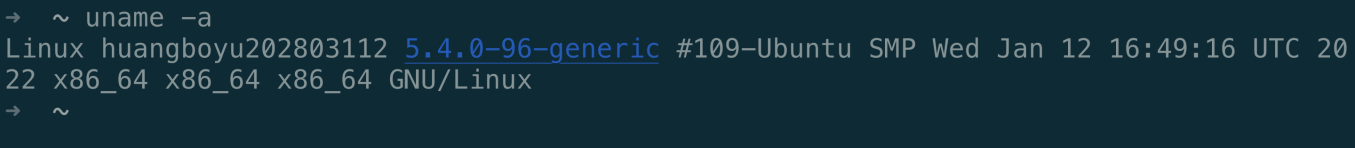
1. Install the Linux kernel
2. Analyze the ext file system source code
3. Analyze the relationship and the structure of the code

# Chapter II Linux Kernel Installation

In this chapter I will explain how to install and run the linux kernel using qemu.

### Experimental environment

uname -a



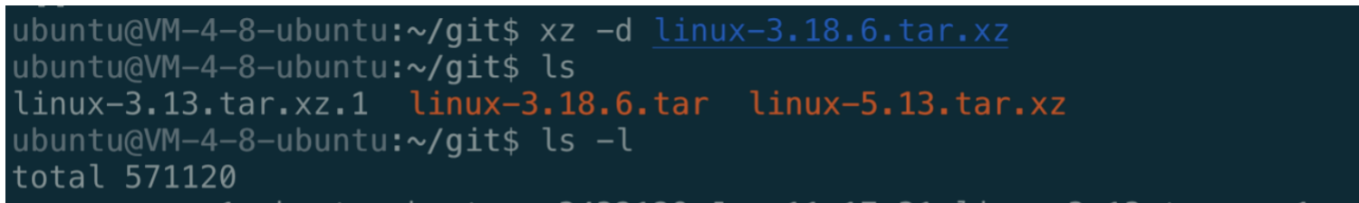
### Experimental steps

Use wget command to download the source code of linux kernel

wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.18.6.tar.xz

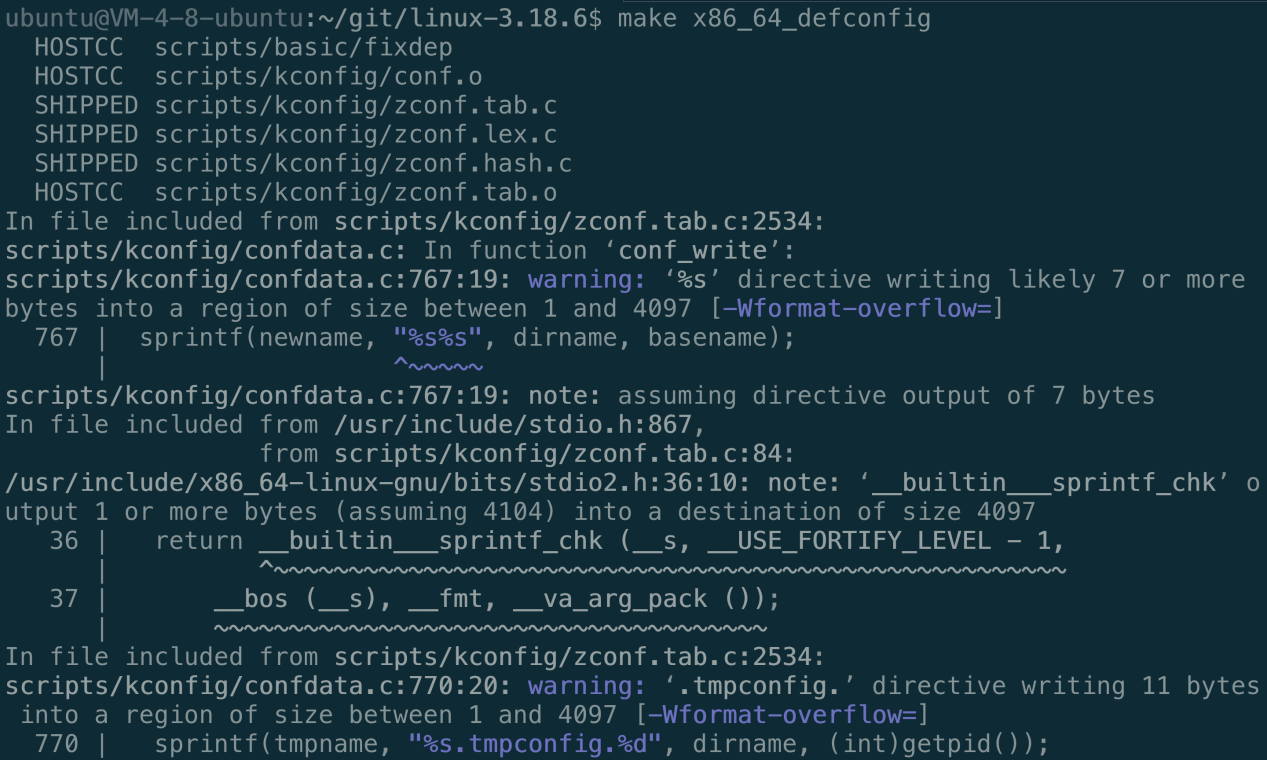


Unzip and extract the tarball

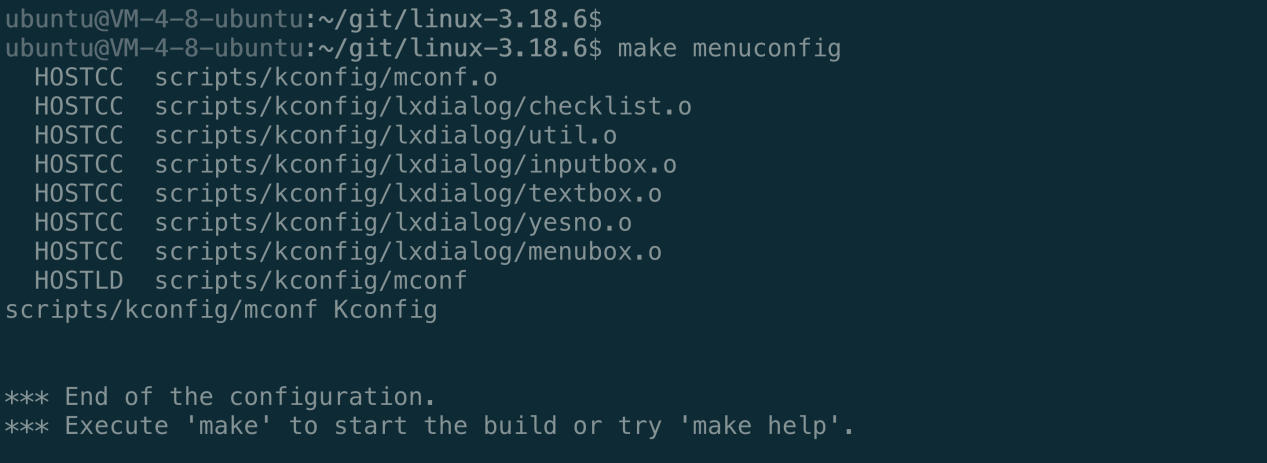


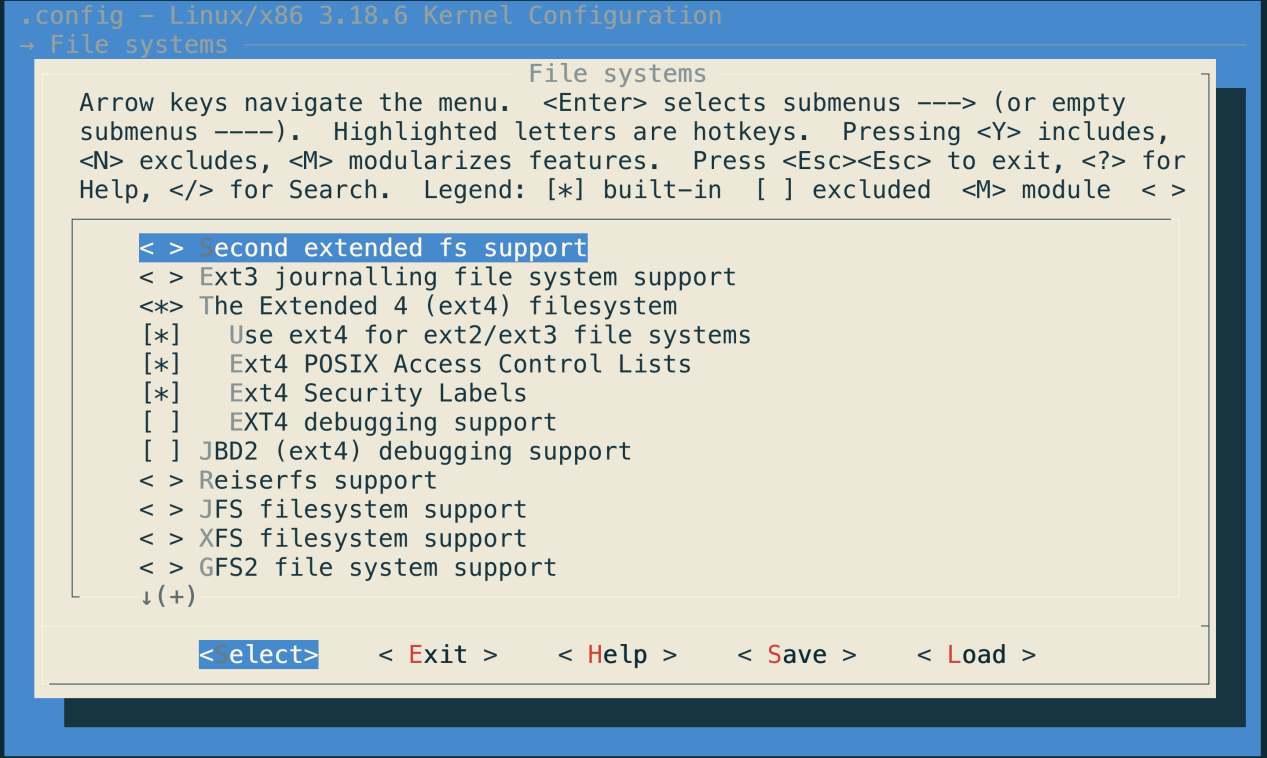
Set up the system envirenment



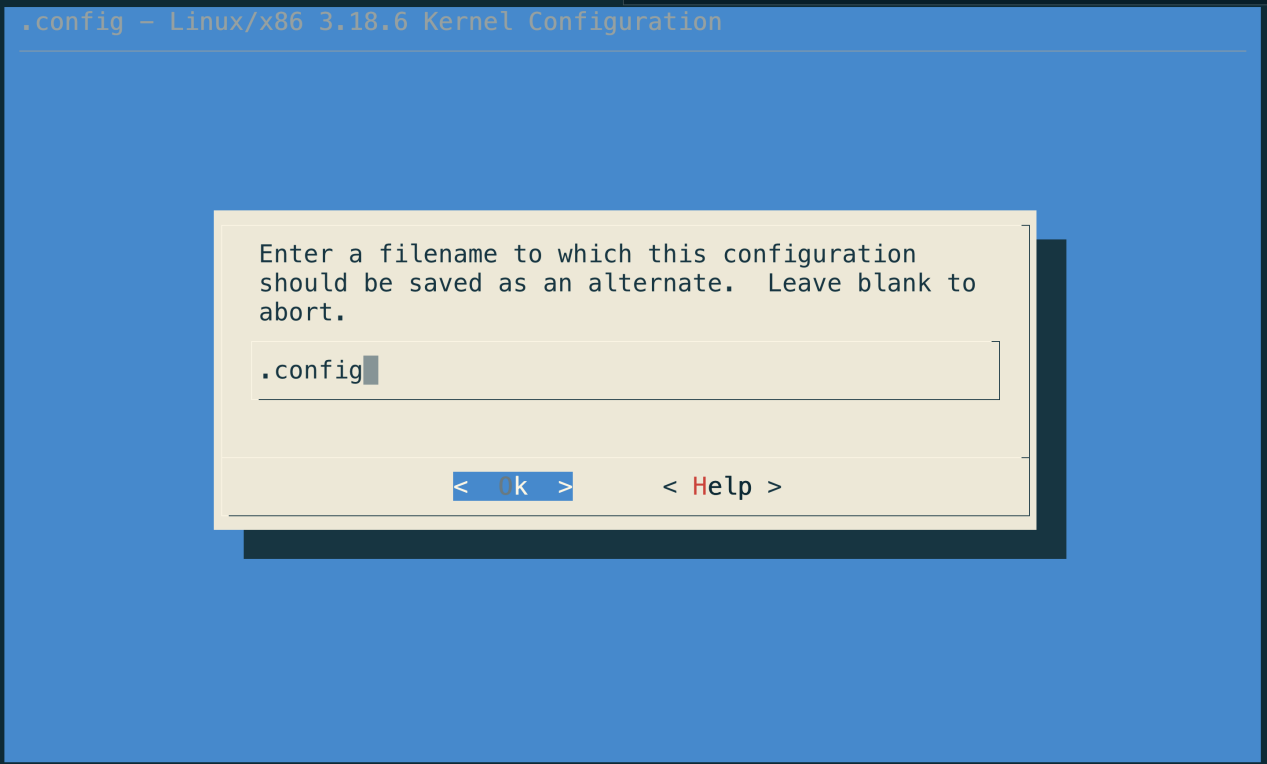


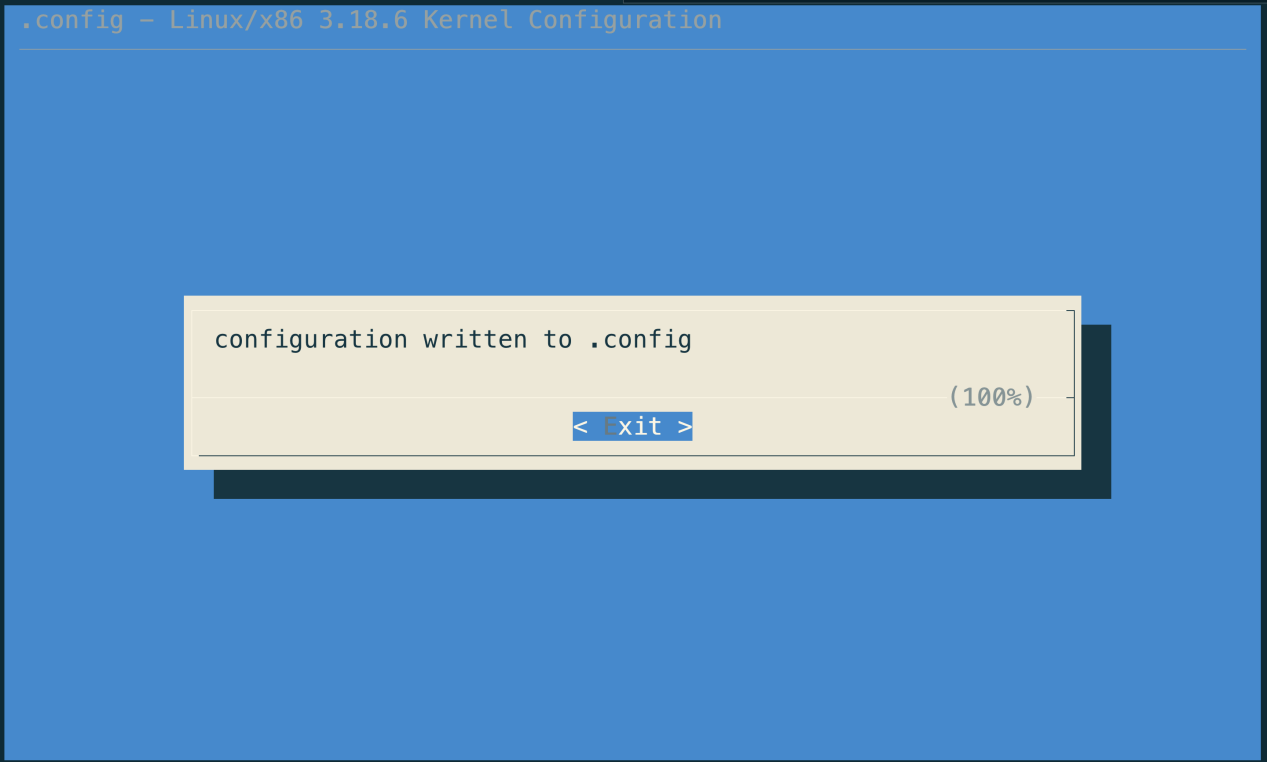
Generate configuration



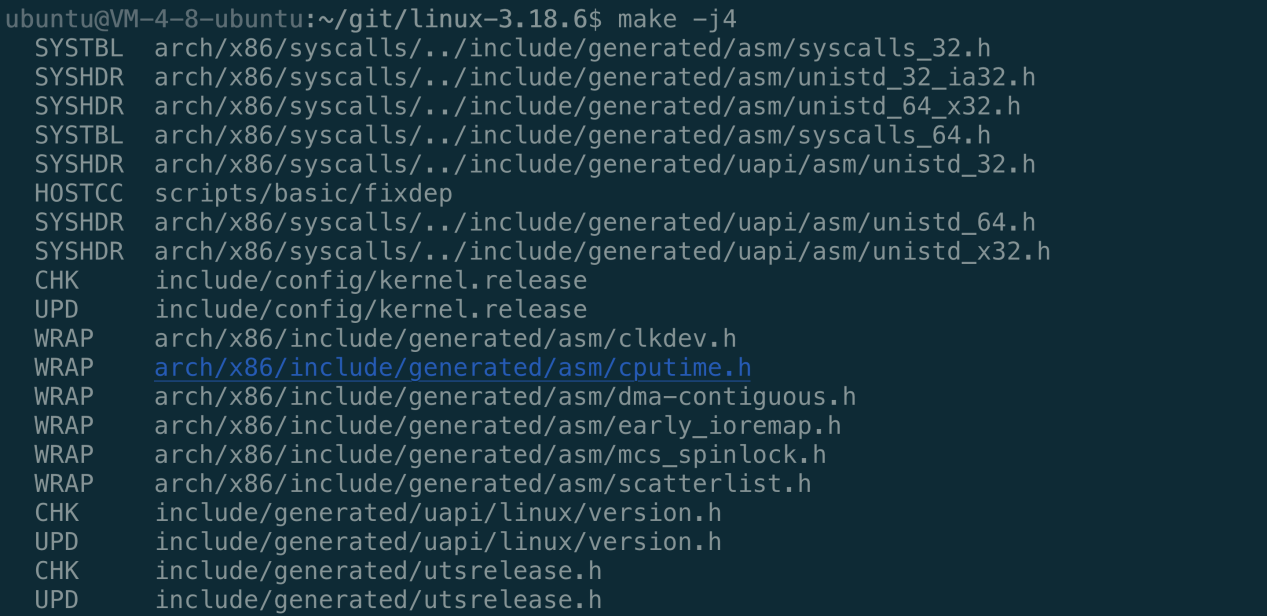


Save to configuration file

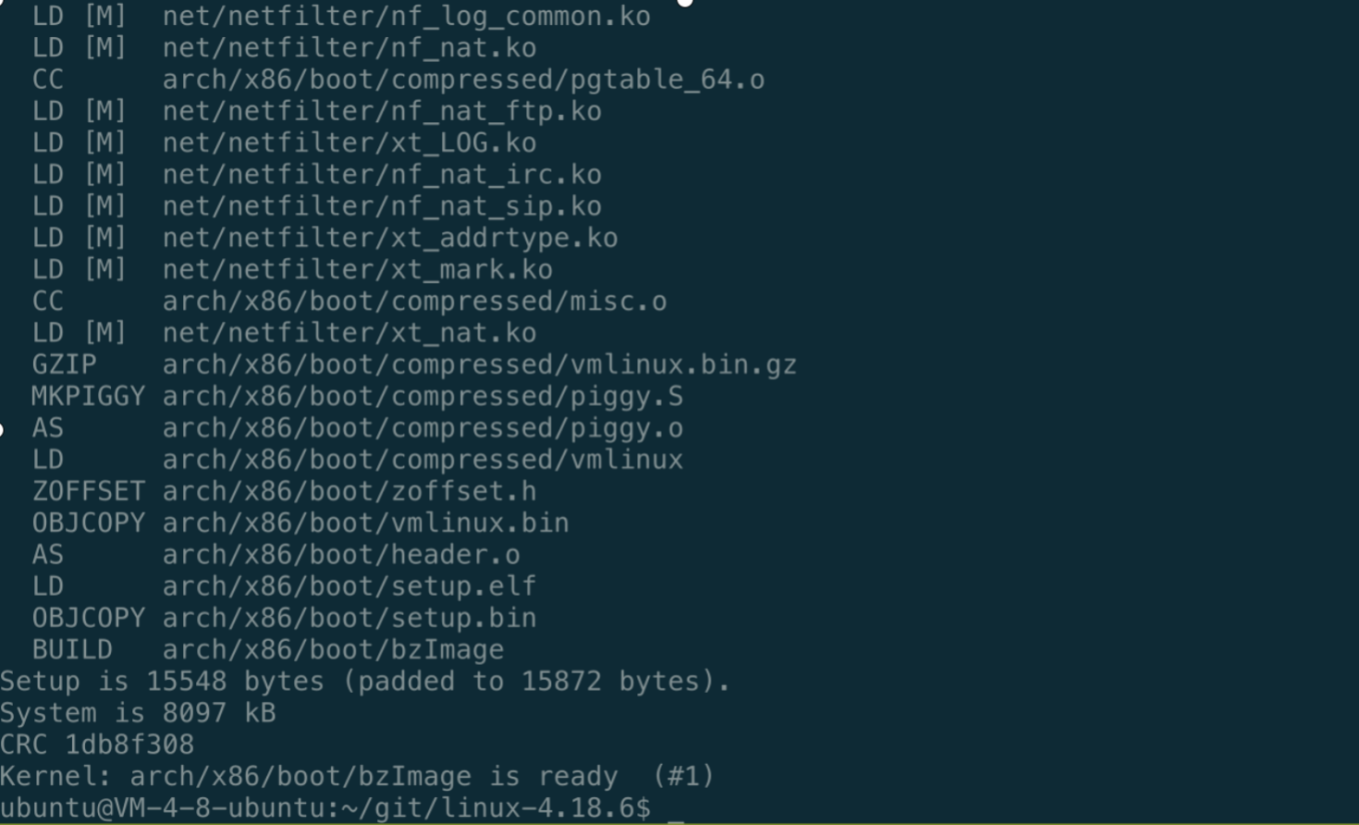




Compile



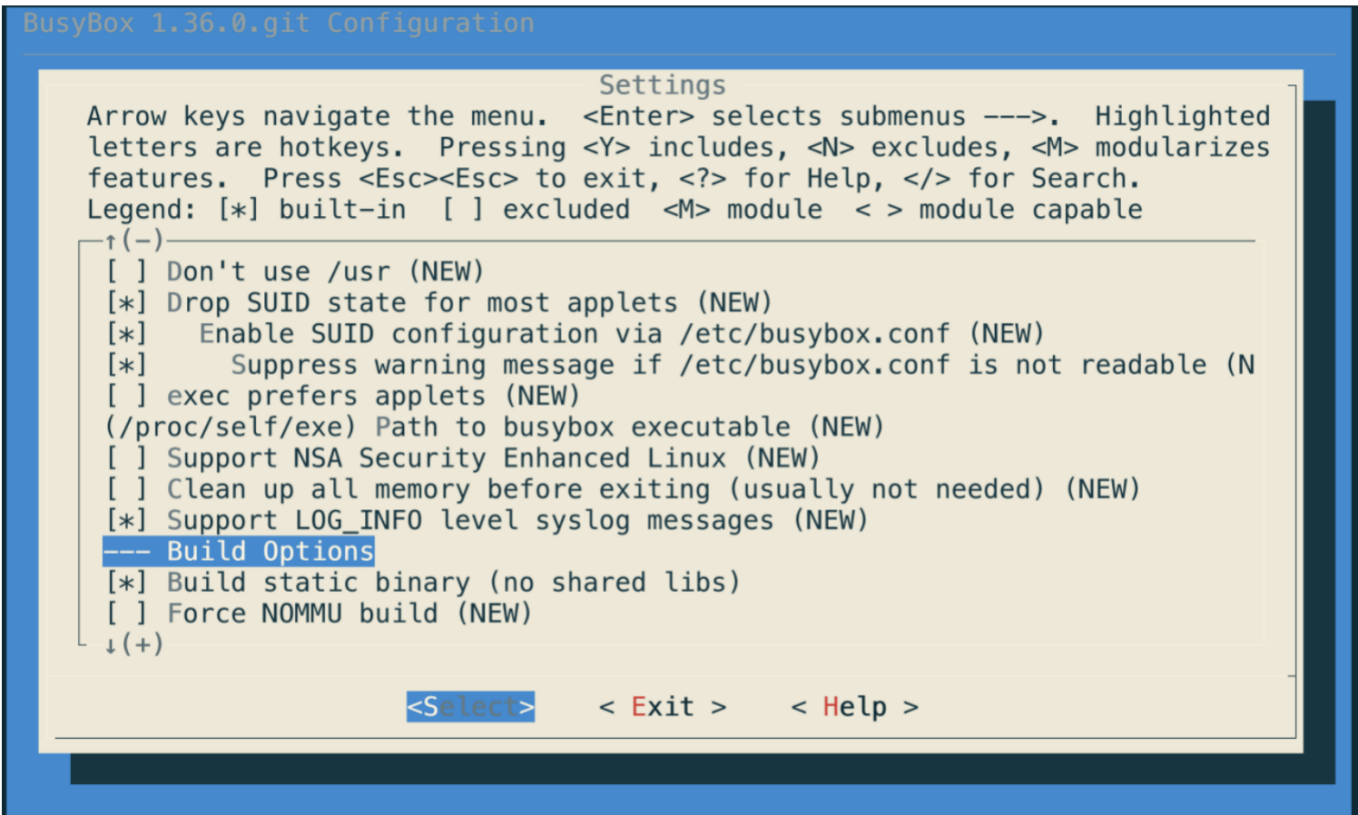
Change kernel version



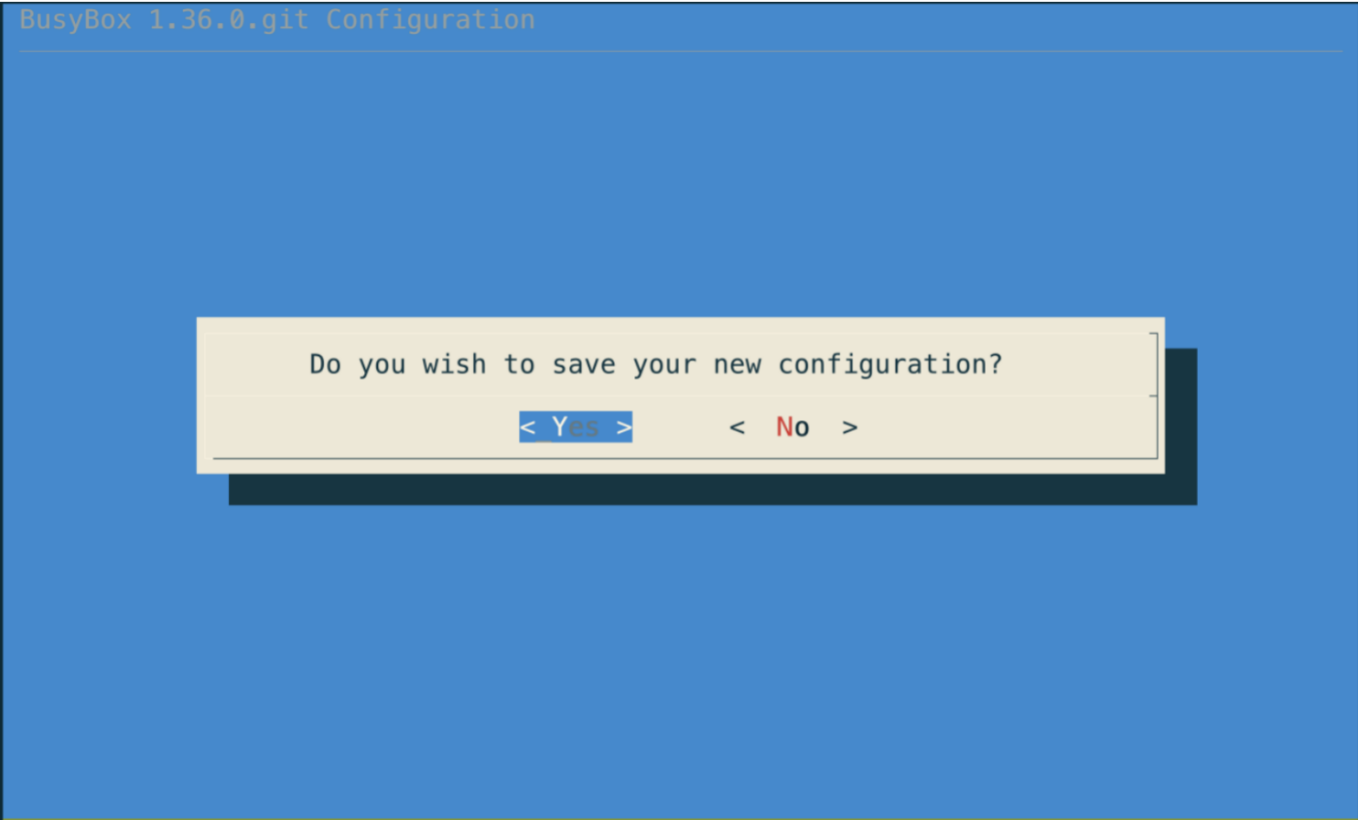
Down load Busybox



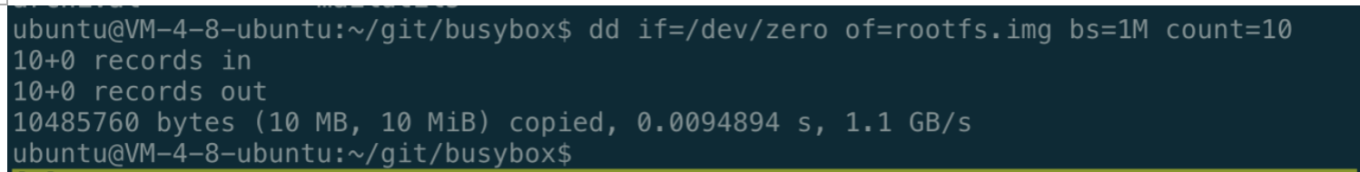
make menuconfig( using static linked)

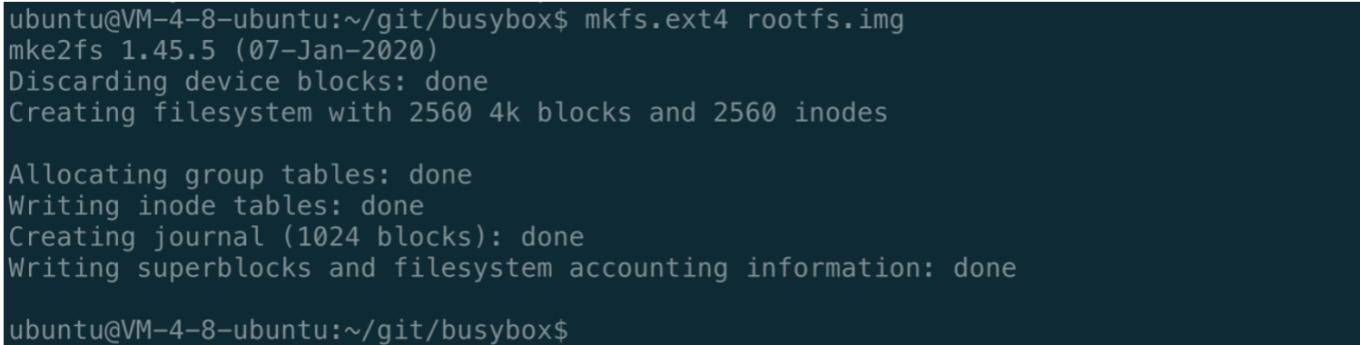


Save configuration

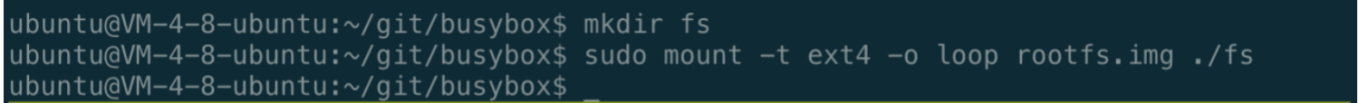


Use dd command to create a disk

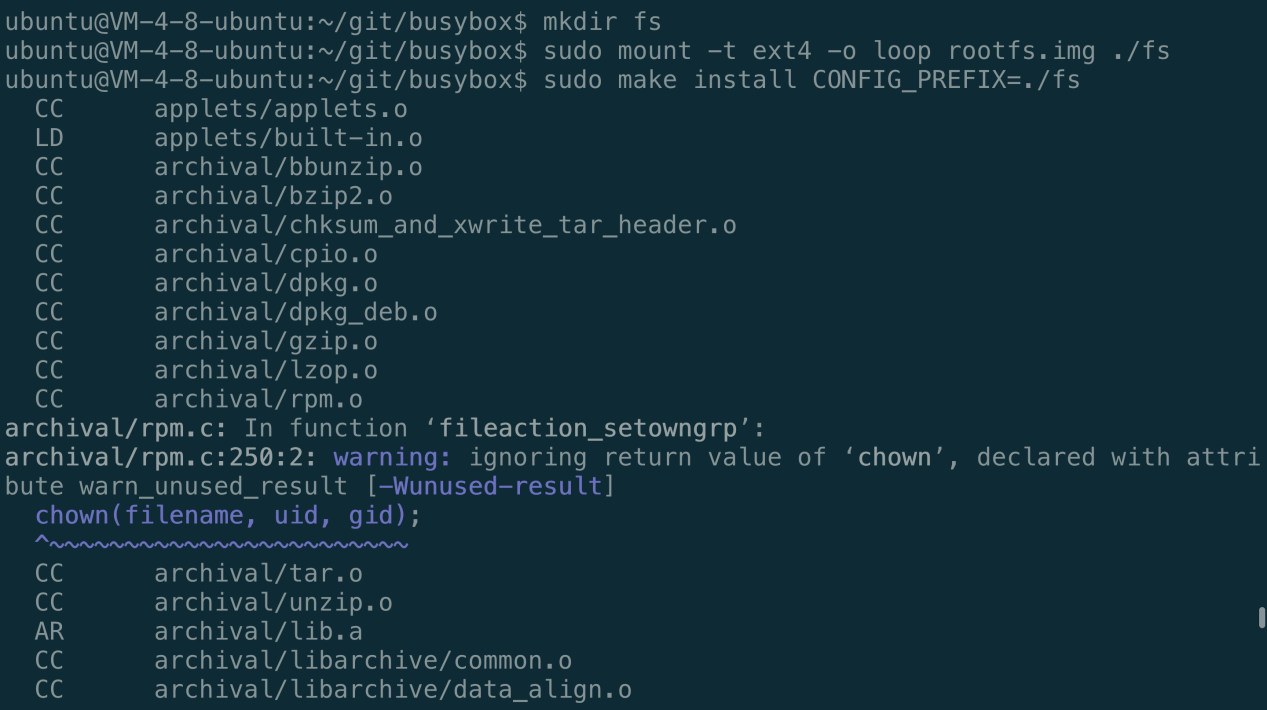


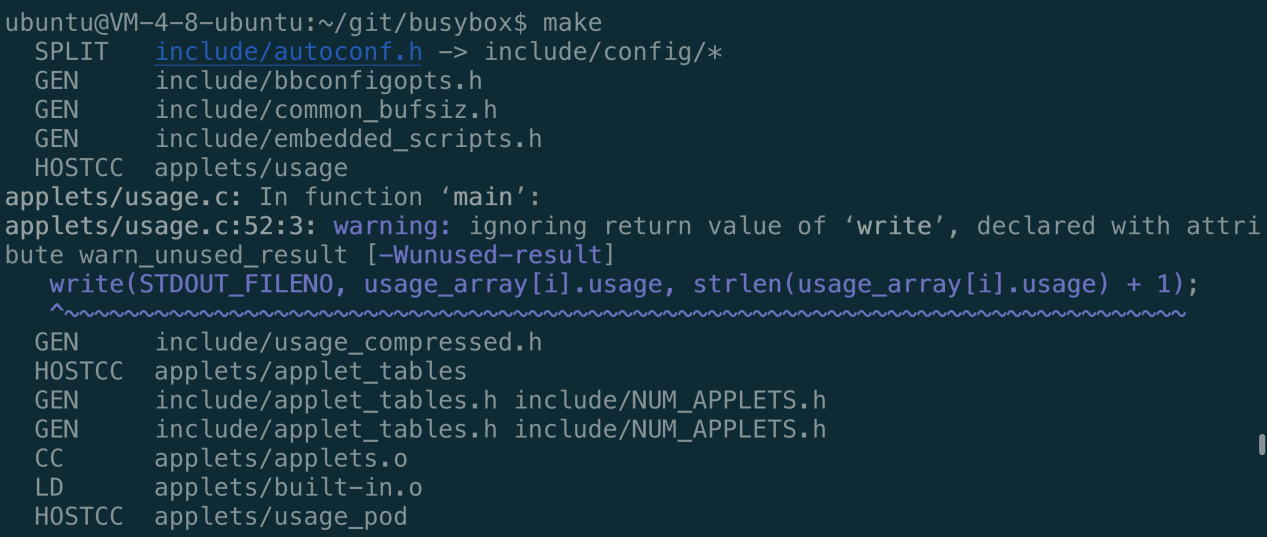


Mout to img

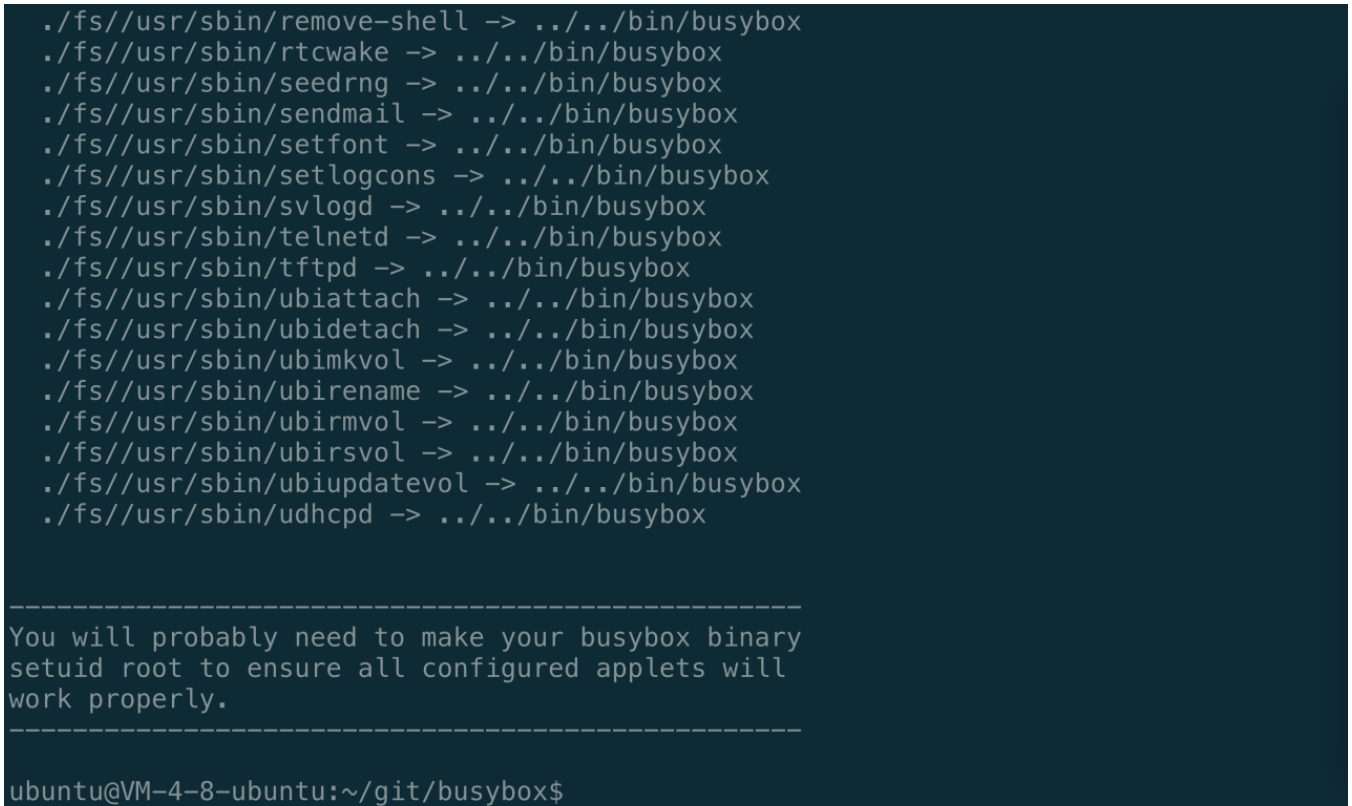


Compile busybox





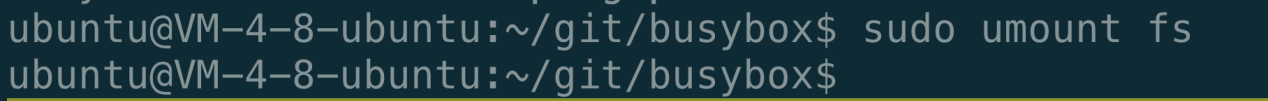
Make succeed



Additional configuration



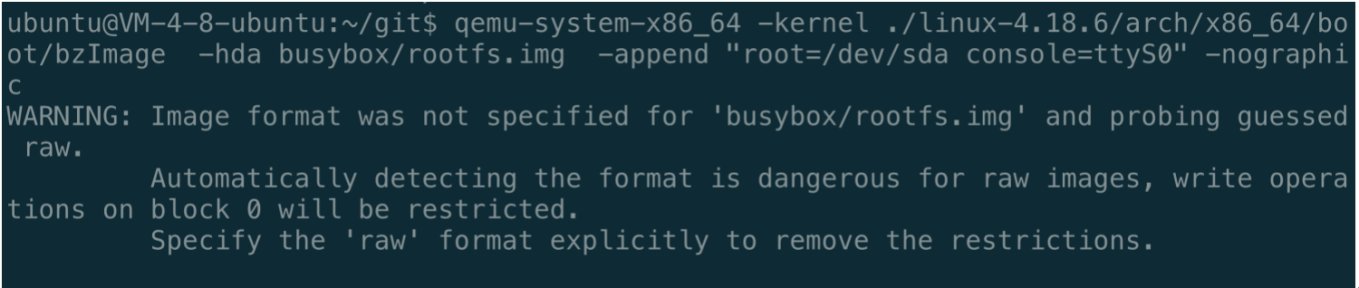
Umount file system



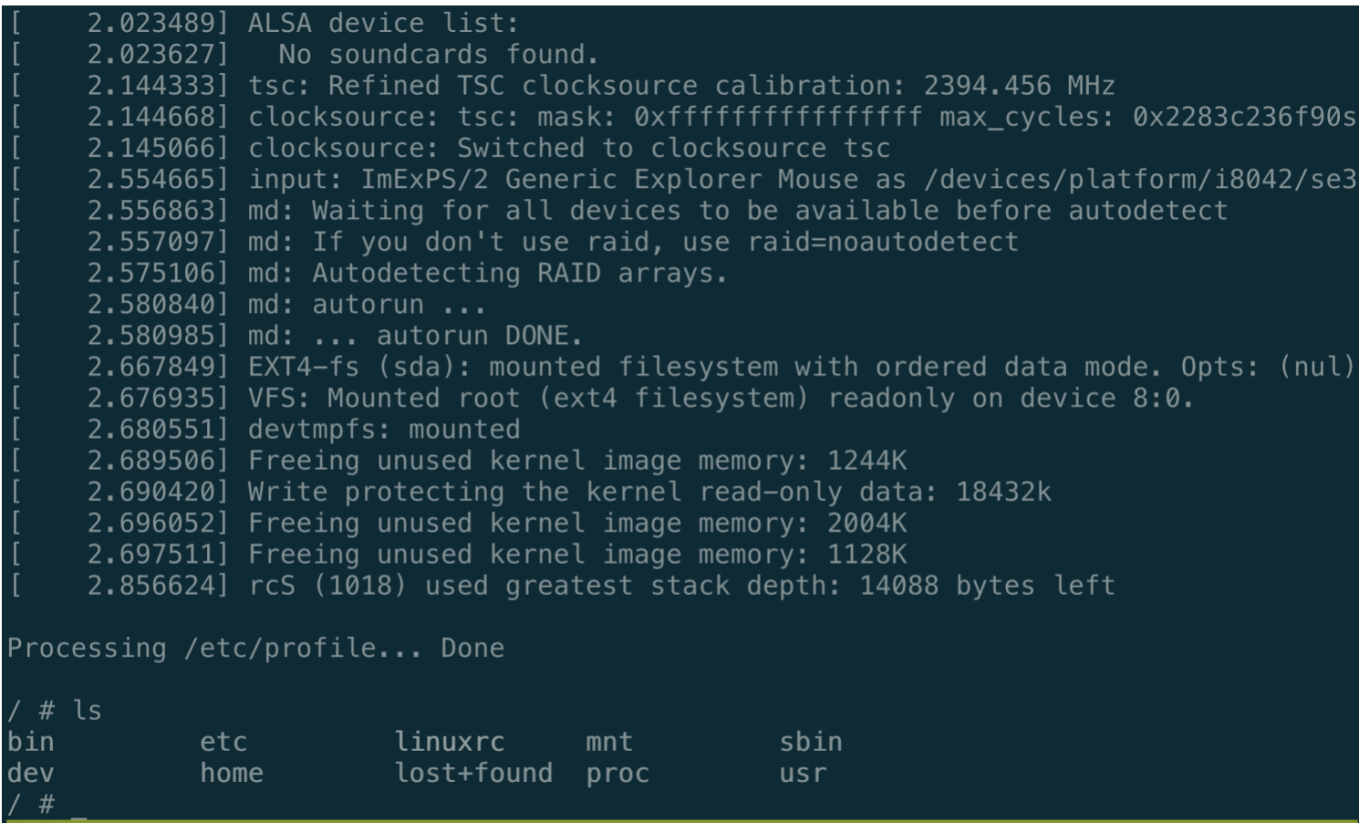
### Start the kernel

Use qemu command to emulate linux system

qemu-system-x86\_64 -kernel ./linux-4.18.6/arch/x86\_64/boot/bzImage -hda busybox/rootfs.img -append "root=/dev/sda rw console=ttyS0" -nographic



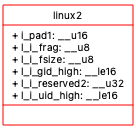
Start kernel succeed



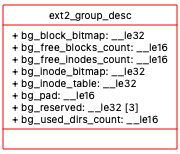
# Chapter III Linux File System Internal

In this chapter I will explain some important code of the Linux.

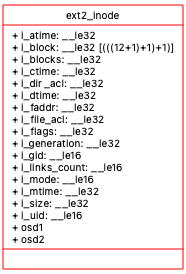
### Sturcture of the EXT2 file system



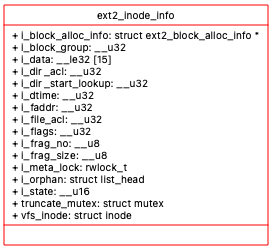
Linux2 structure



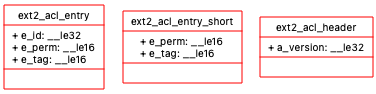
Ext2\_group\_desc



Ext2\_inode

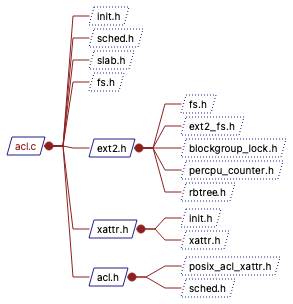


Ext2\_inode\_info



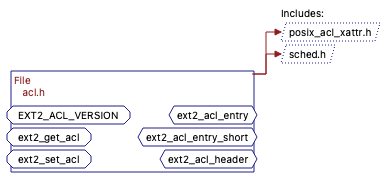
Ext2\_acl structure

### Header dependencies

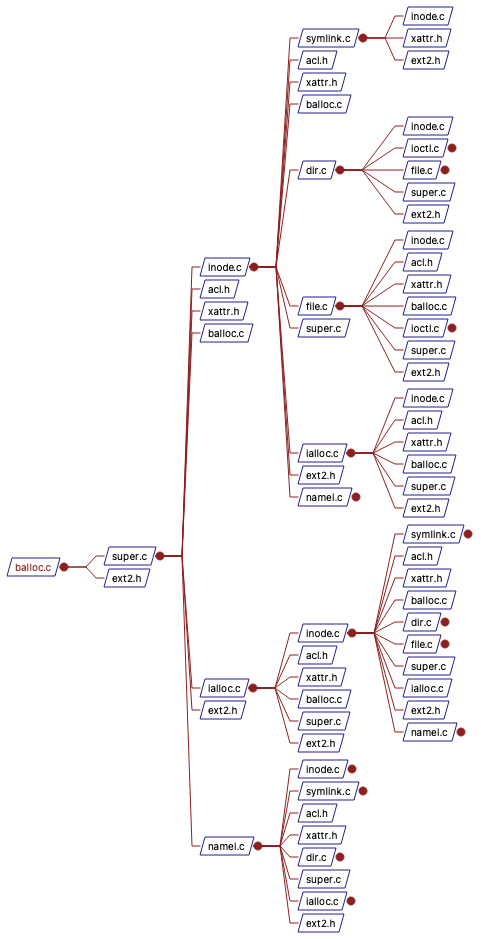




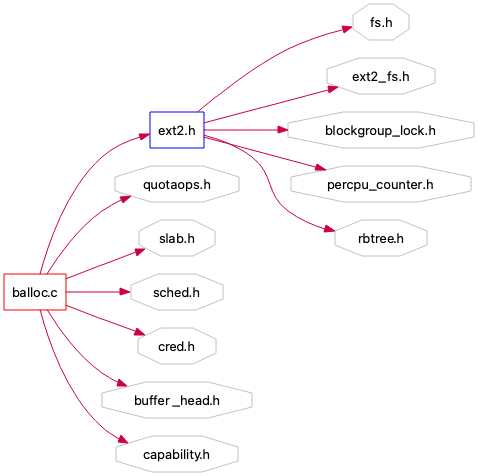
Ext2 acl file dependencies



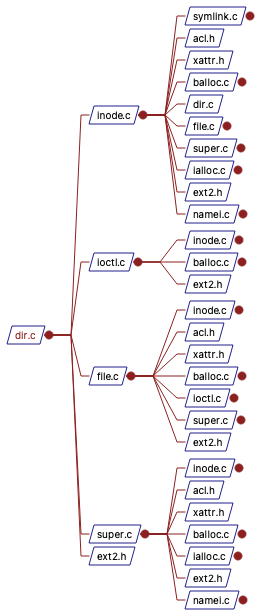
Ext2 functions



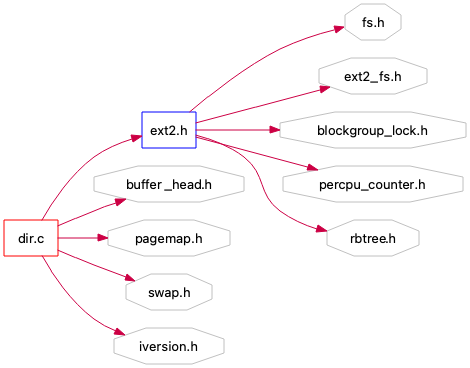
Balloc file dependencies



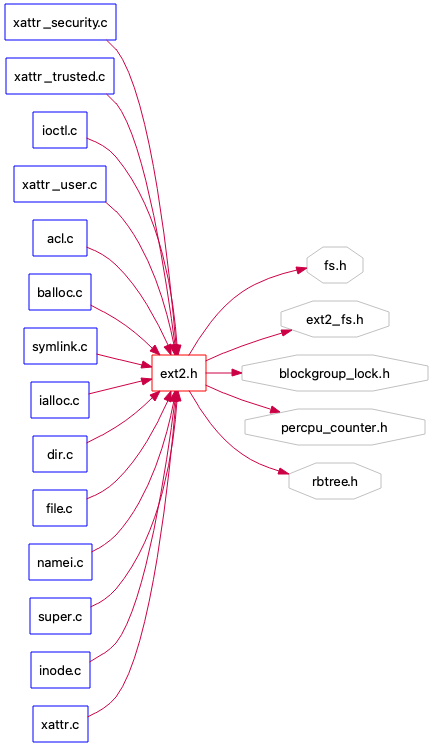
Balloc header dependencies



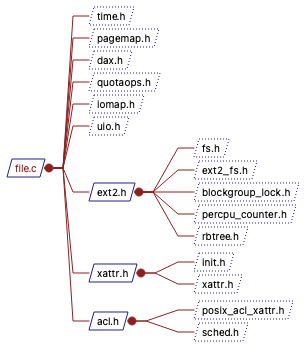
Dir file dependencies



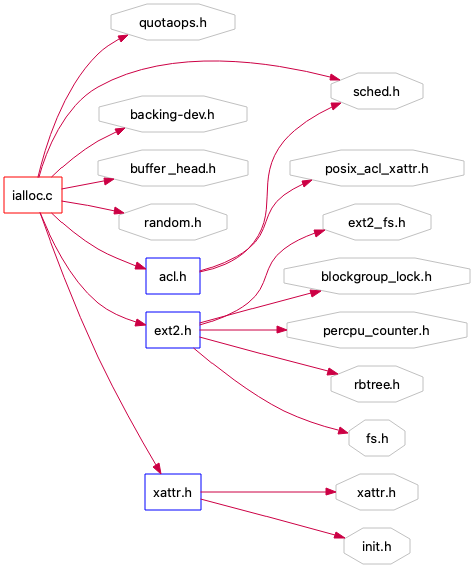
Dir header dependencies



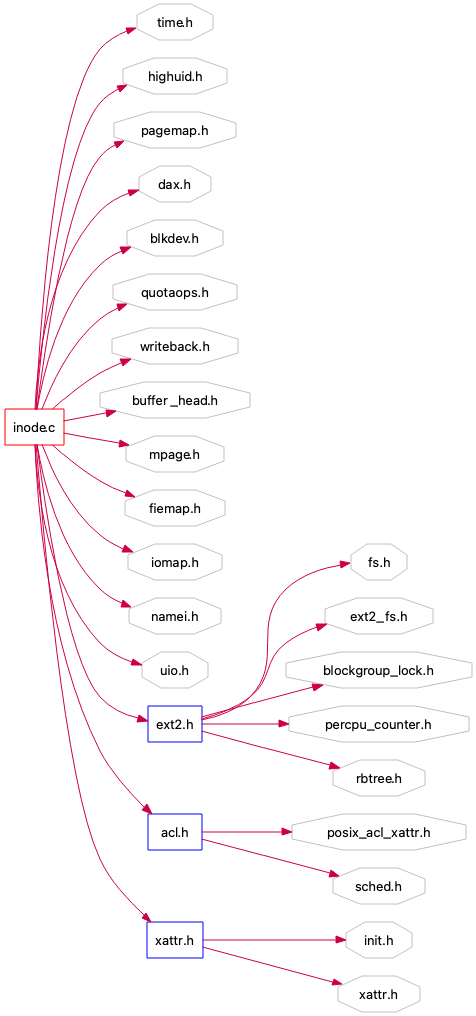
Ext2 file dependencies



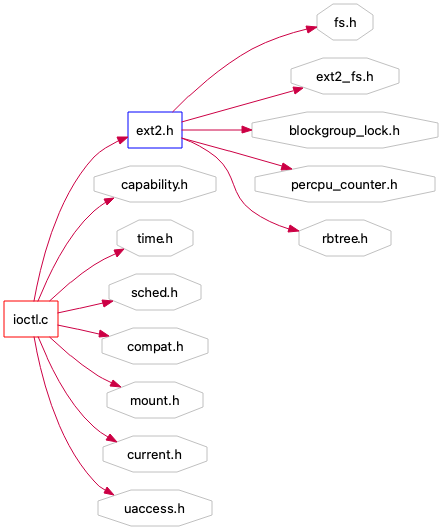
File header dependencies



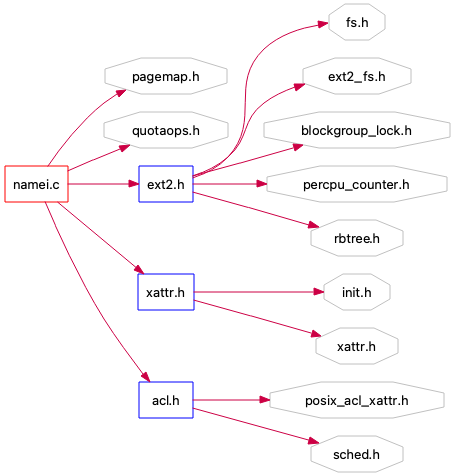
Ialloc header dependencies



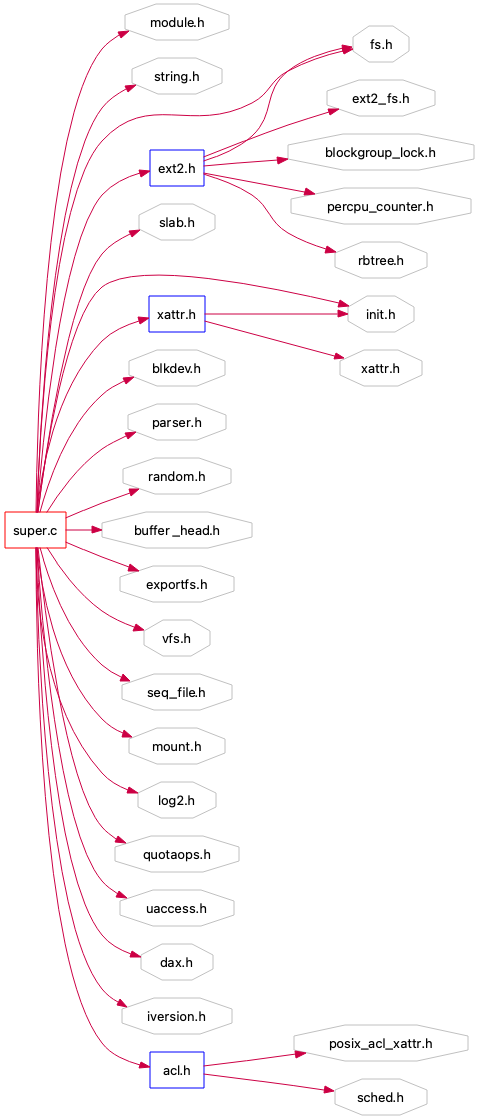
Inode header dependencies



Ioctl header dependencies



Namei header dependencies



Super header dependencies

### Ext2 in Details

The operation command of calling ioctl() expands the conventional system call through the following parameters. Its function is similar to "trapping". All relatively small operations that do not need to set a special system call are classified as IOCTL. Moreover, when adapting the drive of special equipment, it is often achieved by adding new IOCTL commands.

The inode structure on the disk includes the basic information of the file, such as the creator, creation time, etc., the read and write permissions of the file, and the timestamp of the file: CTime refers to the time when the inode was last changed, mtime refers to the time when the file content was last changed, and atime refers to the time when the file was last opened.

An inode points to a file, but multiple file names can point to an inode. The relationship between file names and inodes can be divided into hard link and soft link. Hard link is a backup of the inode of the original file. Deleting one file name will not affect another file name; Soft connection is the path from one file name to another. When the dependent file name is deleted, it will cause access failure.

This is the parameter of mount() system call. Mount is the function of mounting the file system. The file system can be used only when it is mounted.

Each file system has a super block structure. The super block is the place where the file system is used to record and manage information. When the disk file system is mounted on the operating system, the super block will also be read into memory to generate a new super block. Using the super block in memory, you can find the inode corresponding to each file.

Revision level can determine whether the file system supports the features that only exist under this specific revision of the file system, that is, compatibility, and whether it supports new features.

Dentry is a logical attribute of a file. It points to the inode of the file rather than its physical address on the disk. It only exists in memory and is designed to improve the search speed. As shown below, an effective dentry structure must have an inode structure, and all dentry structures form a complete directory tree. ext2\_ dir\_ entry\_ 2 is a newer version.

The information in the inode read from the disk is original and unprocessed, which is called raw\_ inode。 The index node in the memory is divided into two parts. One part belongs to the VFS layer, which is applicable to all file systems; The other part belongs to a specific file system type, as shown below.

In ext2\_ inode\_ I in info structure\_ Data[15] is the most important data, in which some pointers are stored. They directly or indirectly point to all blocks in the disk that store the contents of the file. There are 15 pointers in total. The first 12 are direct pointers, and the last three are "primary indirect pointers", "secondary indirect pointers" and "tertiary indirect pointers" respectively.

struct ext2\_inode\_info {

\_\_le32 i\_data[15];

\_\_u32 i\_flags;

\_\_u32 i\_faddr;

\_\_u8 i\_frag\_no;

\_\_u8 i\_frag\_size;

\_\_u32 i\_file\_acl; \_\_u16 i\_state;

\_\_u32 i\_dir\_acl;

\_\_u32 i\_dtime;

\_\_u32 i\_block\_group;

struct ext2\_block\_alloc\_info \*i\_block\_alloc\_info;

\_\_u32 i\_dir\_start\_lookup;

#ifdef CONFIG\_EXT2\_FS\_XATTR

struct rw\_semaphore xattr\_sem;

#endif

rwlock\_t i\_meta\_lock;

struct mutex truncate\_mutex;

struct inode vfs\_inode;

struct list\_head i\_orphan;

#ifdef CONFIG\_QUOTA

struct dquot \*i\_dquot[MAXQUOTAS];

#endif

};

ext2\_fs.h

struct ext2\_group\_desc

{

\_\_le32 bg\_block\_bitmap; /\* Blocks bitmap block \*/

\_\_le32 bg\_inode\_bitmap; /\* Inodes bitmap block \*/

\_\_le32 bg\_inode\_table; /\* Inodes table block \*/

\_\_le16 bg\_free\_blocks\_count; /\* Free blocks count \*/

\_\_le16 bg\_free\_inodes\_count; /\* Free inodes count \*/

\_\_le16 bg\_used\_dirs\_count; /\* Directories count \*/

\_\_le16 bg\_pad;

\_\_le32 bg\_reserved[3];

};

/\*

Each block group contains many group descriptors, and each block group in the file system corresponds to a copy of the group descriptor. Therefore, from each block group, the following information of all other block groups in the system can be determined:

Location of blocks and inode bitmaps

Location of inode table

Number of free blocks and inodes

\*/

<ext2\_fs.h>

struct ext2\_dir\_entry\_2 {

\_\_le32 inode; /\* Inode number \*/

\_\_le16 rec\_len; /\* Directory entry length \*/ //Offset, indicating from rec\_ Len field end to next rec\_ Offset at the end of len field

\_\_u8 name\_len; /\* Name length \*/

\_\_u8 file\_type;

char name[EXT2\_NAME\_LEN]; /\* File name \*/

};

enum {// FILE type

EXT2\_FT\_UNKNOWN,

EXT2\_FT\_REG\_FILE,

EXT2\_FT\_DIR,

EXT2\_FT\_CHRDEV,

EXT2\_FT\_BLKDEV,

EXT2\_FT\_FIFO,

EXT2\_FT\_SOCK,

EXT2\_FT\_SYMLINK,

EXT2\_FT\_MAX

};

/\*

The type of file is not defined in the inode itself, but in the file of the corresponding directory item\_ Type field. But for different file types, the contents of inode will be different. It should be noted that only directories and ordinary files occupy data blocks of the hard disk, and all other types can be fully described using the information in inode.

\*/

/\*

The following codes are prototypes of specific file operation methods, such as inode reading and writing, new node allocation, and so on. Then the specific implementation method is analyzed.

\*/

extern int ext2\_bg\_has\_super(struct super\_block \*sb, int group);

extern unsigned long ext2\_bg\_num\_gdb(struct super\_block \*sb, int group);

extern ext2\_fsblk\_t ext2\_new\_block(struct inode \*, unsigned long, int \*);

extern ext2\_fsblk\_t ext2\_new\_blocks(struct inode \*, unsigned long,

unsigned long \*, int \*);

extern int ext2\_data\_block\_valid(struct ext2\_sb\_info \*sbi, ext2\_fsblk\_t start\_blk,

unsigned int count);

# Conclusion

## Fat32

FAT32 is probably the most compatible file system since it is compatible with almost all operating systems, including all versions of Windows starting with Windows 95. Other operating systems such as Mac OSX, Linux, Unix also support this file system.

FAT32 has its limitations, such as it supports a file size of up to 4 GB and a volume size of 8 TB. If there are a large number of files on a single volume, its performance will also decrease.

## ExFat

ExFAT is introduced to overcome the limitations of FAT32 file system. Although everything else is similar to FAT32, the maximum file size of exFAT file system is 16 EB (Exabyte) and the maximum volume size is 24 ZB (zettabyte).

## Ext

The first generation extended file system, published in April, 1992, is the first file system for the Linux core. Adopt the metadata structure of UNIX file system (UFS) to overcome the poor performance of MINIX file system.

It is the first file system implemented with virtual file system on Linux. To vercome the poor performance of MINIX file system.

## Ext2

The second generation extended file system is the file system used by the Linux kernel. It was initially designed by R é my card to replace ext and was added to Linux core support in January 1993. The classic implementation of ext2 is the Ext2fs file system driver in the Linux kernel, which can support 2TB file system at most. When the Linux core version 2.6 is expanded, it can support 32tb.

In EXT2 file system, the file is uniquely identified by inode (which contains all the information of the file). A file may correspond to multiple file names. The file will not be deleted until all file names are deleted. In addition, when the same file is stored on disk and opened, the corresponding inode is different, and the kernel is responsible for synchronization.

## Ext3

Ext3 is the third generation extended file system (English: third extended file system, abbreviated as ext3).

Ext3 file system is developed directly from EXT2 file system. At present, ext3 file system is very stable and reliable. It is fully compatible with EXT2 file system. Users can smoothly transition to a file system with sound logging functions.

1. High availability: after the ext3 file system is used, the system does not need to check the file system even after an abnormal shutdown.
2. Data integrity: it avoids the damage to the file system caused by unexpected downtime.
3. Speed of file system: because the logging function of ext3 optimizes the drive read-write head of the disk. Therefore, the reading and writing performance of the file system is not lower than that of the EXT2 file system.
4. 4. Data conversion: "it is very easy to convert from EXT2 file system to ext3 file system.
5. Multiple logging modes.
6. Very stable, journal support, up to 16TB, supported by all GNU/Linux distributions. Ext3 logs are not flash friendly. It does help with quick recovery though. It is simple and powerful which support most version of Linux, so I choose it.

## Ext4

Ext4 is the fourth generation extended file system, which is the log file system under Linux system and the successor version of ext3 file system. Ext4 was implemented by the development team led by Theodore Tso, the maintainer of ext3, and was introduced into the linux2.6.19 kernel.

Ext4 is an improved version of ext3, which modifies some important data structures in ext3, not just like ext3 to ext2, but also adds a logging function. Ext4 can provide better performance, reliability and richer functions

Stable in newer kernel versions, faster filesystem checks, improved large file performance (extended support), journaled checksums. RHEL only announced stable Ext4 support in version 5.6. The capacity of e2fsprogs is still limited to 16TB.

1. Compatible with ext3: by executing several commands, you can migrate from ext3 to ext4 online without reformatting the disk or reinstalling the system.
2. Larger file systems and larger files: compared with the maximum 16TB file system and the maximum 2TB file system currently supported by ext3, ext4 supports 1eb (1048576tb, 1eb=1024pb, 1pb=1024tb) file systems and 16TB files respectively. Unlimited number of subdirectories: ext3 currently only supports 32000 subdirectories, while ext4 supports an unlimited number of subdirectories.
3. Extensions: ext4 introduces the popular concept of extensions in modern file systems. Each extension is a group of continuous data blocks. Compared with ext3, it uses indirect block mapping, which improves a lot of efficiency.
4. Multi block allocation: the multi block allocator (mballoc) of ext4 supports the allocation of multiple data blocks at a time\* 6. Delayed allocation 7. Fast fsck 8. Log verification 9. "No journaling" mode 10. Online defragmentation 11. Inode related features: compared with ext3's default inode size of 128 bytes, ext4's default inode size is 256 bytes.