Unfs分析

Q.L.

2013-8-1

目录

[1. 主要流程 5](#_Toc366092515)

[2. 主要的数据结构 9](#_Toc366092516)

[2.1. NFS的常数和文件模式 9](#_Toc366092517)

[2.2. NFS的调用返回值 10](#_Toc366092518)

[2.3. NFS的文件类型： 11](#_Toc366092519)

[2.4. NFS的文件句柄： 12](#_Toc366092520)

[2.5. NFS的文件属性结构： 12](#_Toc366092521)

[2.6. Mount的常用返回值： 13](#_Toc366092522)

[1. Daemon.c 13](#_Toc366092523)

[1.1. main（） 14](#_Toc366092524)

[1.1.1. 流程 14](#_Toc366092525)

[1.1.2. 核心代码 14](#_Toc366092526)

[1.2. parse\_options() 15](#_Toc366092527)

[1.2.1. 使用说明 15](#_Toc366092528)

[1.3. register\_nfs\_service()： 15](#_Toc366092529)

[1.3.1. 使用说明 15](#_Toc366092530)

[1.3.2. 核心代码 15](#_Toc366092531)

[1.4. nfs3\_program\_3 ()： 16](#_Toc366092532)

[1.4.1. 使用说明 16](#_Toc366092533)

[1.4.2. 核心代码 16](#_Toc366092534)

[1.5. register\_mount\_service() 17](#_Toc366092535)

[1.5.1. 使用说明 17](#_Toc366092536)

[1.5.2. 核心代码 17](#_Toc366092537)

[1.6. mountprog\_3() 18](#_Toc366092538)

[1.6.1. 使用说明 18](#_Toc366092539)

[1.6.2. 核心代码 18](#_Toc366092540)

[1.7. unfs3\_svc\_run() 18](#_Toc366092541)

[1.7.1. 使用说明 18](#_Toc366092542)

[1.7.2. 核心代码 19](#_Toc366092543)

[2. mount.c 19](#_Toc366092544)

[2.1. 重构mountres3 \* mountproc\_mnt\_3\_svc(dirpath \*, struct svc\_req \*); 19](#_Toc366092545)

[2.1.1. 使用说明 19](#_Toc366092546)

[2.1.2. 核心代码 20](#_Toc366092547)

[2.1.3. 代码重构 20](#_Toc366092548)

[2.2. 重构void \* mountproc\_umnt\_3\_svc(dirpath \*, struct svc\_req \*); 20](#_Toc366092549)

[2.2.1. 使用说明 20](#_Toc366092550)

[2.2.2. 核心代码 21](#_Toc366092551)

[2.2.3. 代码重构 21](#_Toc366092552)

[2.3. void \* mountproc\_umntall\_3\_svc(void \*, struct svc\_req \*); 21](#_Toc366092553)

[2.3.1. 使用说明 21](#_Toc366092554)

[2.3.2. 核心代码 21](#_Toc366092555)

[2.3.3. 代码重构 21](#_Toc366092556)

[2.4. void \* mountproc\_null\_3\_svc(void \*, struct svc\_req \*) 22](#_Toc366092557)

[2.4.1. 使用说明 22](#_Toc366092558)

[2.4.2. 代码重构 22](#_Toc366092559)

[2.5. exports \* mountproc\_export\_3\_svc(void \*, struct svc\_req \*); 22](#_Toc366092560)

[2.5.1. 使用说明 22](#_Toc366092561)

[2.5.2. 代码重构 22](#_Toc366092562)

[2.6. mountlist \* mountproc\_dump\_3\_svc(void \*, struct svc\_req \*); 22](#_Toc366092563)

[2.6.1. 使用说明 22](#_Toc366092564)

[2.6.2. 代码重构 22](#_Toc366092565)

[2.7. static void add\_mount(const char \*path, struct svc\_req \*rqstp) 23](#_Toc366092566)

[2.7.1. 使用说明 23](#_Toc366092567)

[2.7.2. 代码重构 23](#_Toc366092568)

[2.8. static void remove\_mount(const char \*path, struct svc\_req \*rqstp) 23](#_Toc366092569)

[2.8.1. 使用说明 23](#_Toc366092570)

[2.8.2. 代码重构 23](#_Toc366092571)

[3. nfs.c 24](#_Toc366092572)

[3.1. void \*nfsproc3\_null\_3\_svc(U(void \*argp), U(struct svc\_req \*rqstp)) 24](#_Toc366092573)

[3.1.1. 使用说明 24](#_Toc366092574)

[3.2. FSINFO3res \*nfsproc3\_fsinfo\_3\_svc(FSINFO3args \* argp, struct svc\_req \* rqstp) 24](#_Toc366092575)

[3.2.1. 使用说明 24](#_Toc366092576)

[3.2.2. 核心代码 24](#_Toc366092577)

[3.2.3. 重构内容 24](#_Toc366092578)

[3.3. PATHCONF3res \*nfsproc3\_pathconf\_3\_svc(PATHCONF3args \* argp, struct svc\_req \* rqstp) 25](#_Toc366092579)

[3.3.1. 使用说明 25](#_Toc366092580)

[3.3.2. 核心代码 25](#_Toc366092581)

[3.3.3. 重构内容 25](#_Toc366092582)

[3.4. ACCESS3res \* nfsproc3\_access\_3\_svc(ACCESS3args \*, struct svc\_req \*); 25](#_Toc366092583)

[3.4.1. 使用说明 25](#_Toc366092584)

[3.4.2. 核心代码 25](#_Toc366092585)

[3.4.3. 重构内容 26](#_Toc366092586)

[3.5. GETATTR3res \*nfsproc3\_getattr\_3\_svc(GETATTR3args \* argp, struct svc\_req \* rqstp) 26](#_Toc366092587)

[3.5.1. 使用说明 26](#_Toc366092588)

[3.5.2. 核心代码 28](#_Toc366092589)

[3.5.3. 重构内容 28](#_Toc366092590)

[3.6. READDIR3res \* nfsproc3\_readdir\_3\_svc(READDIR3args \*, struct svc\_req \*); 28](#_Toc366092591)

[3.6.1. 使用说明 28](#_Toc366092592)

[3.6.2. 核心代码 28](#_Toc366092593)

[3.6.3. 重构内容 28](#_Toc366092594)

[3.7. READDIRPLUS3res \* nfsproc3\_readdirplus\_3\_svc(READDIRPLUS3args \*, struct svc\_req \*); 29](#_Toc366092595)

[3.7.1. 使用说明 29](#_Toc366092596)

[3.7.2. 重构内容 29](#_Toc366092597)

[3.8. READ3res \* nfsproc3\_read\_3\_svc(READ3args \*, struct svc\_req \*); 30](#_Toc366092598)

[3.9. READLINK3res \* nfsproc3\_readlink\_3\_svc(READLINK3args \*, struct svc\_req \*); 30](#_Toc366092599)

[3.10. WRITE3res \* nfsproc3\_write\_3\_svc(WRITE3args \*, struct svc\_req \*); 30](#_Toc366092600)

[3.11. CREATE3res \* nfsproc3\_create\_3\_svc(CREATE3args \*, struct svc\_req \*); 30](#_Toc366092601)

[3.12. MKDIR3res \* nfsproc3\_mkdir\_3\_svc(MKDIR3args \*, struct svc\_req \*); 30](#_Toc366092602)

[3.13. SYMLINK3res \* nfsproc3\_symlink\_3\_svc(SYMLINK3args \*, struct svc\_req \*); 30](#_Toc366092603)

[3.14. MKNOD3res \* nfsproc3\_mknod\_3\_svc(MKNOD3args \*, struct svc\_req \*); 30](#_Toc366092604)

[3.15. LOOKUP3res \* nfsproc3\_lookup\_3\_svc(LOOKUP3args \*, struct svc\_req \*); 30](#_Toc366092605)

[3.16. REMOVE3res \* nfsproc3\_remove\_3\_svc(REMOVE3args \*, struct svc\_req \*); 31](#_Toc366092606)

[3.17. RMDIR3res \* nfsproc3\_rmdir\_3\_svc(RMDIR3args \*, struct svc\_req \*); 31](#_Toc366092607)

[3.18. RENAME3res \* nfsproc3\_rename\_3\_svc(RENAME3args \*, struct svc\_req \*); 31](#_Toc366092608)

[3.19. LINK3res \* nfsproc3\_link\_3\_svc(LINK3args \*, struct svc\_req \*); 31](#_Toc366092609)

[3.20. FSSTAT3res \* nfsproc3\_fsstat\_3\_svc(FSSTAT3args \*, struct svc\_req \*); 31](#_Toc366092610)

[3.21. SETATTR3res \* nfsproc3\_setattr\_3\_svc(SETATTR3args \*, struct svc\_req \*); 31](#_Toc366092611)

[3.22. COMMIT3res \* nfsproc3\_commit\_3\_svc(COMMIT3args \*, struct svc\_req \*); 31](#_Toc366092612)

[3.23. int nfs3\_program\_3\_freeresult (SVCXPRT \*, xdrproc\_t, caddr\_t); 31](#_Toc366092613)

[4. fh.c 31](#_Toc366092614)

[4.1. static int fh\_rec(const unfs3\_fh\_t \* fh, int pos, const char \*lead,char \*result) 32](#_Toc366092615)

[4.1.1. 使用说明 32](#_Toc366092616)

[4.1.2. 核心代码结构 32](#_Toc366092617)

[4.2. char \*fh\_decomp\_raw(const unfs3\_fh\_t \*fh); 32](#_Toc366092618)

[4.2.1. 使用说明 32](#_Toc366092619)

[4.2.2. 核心代码结构 32](#_Toc366092620)

[4.3. unfs3\_fh\_t fh\_comp\_raw(const char \*path, struct svc\_req \*rqstp, int need\_dir); 33](#_Toc366092621)

[4.3.1. 使用说明 33](#_Toc366092622)

[4.3.2. 原核心代码结构 33](#_Toc366092623)

[4.4. uint32 get\_gen(backend\_statstruct obuf, int fd, const char \*path); 34](#_Toc366092624)

[4.4.1. 使用说明 34](#_Toc366092625)

[4.5. unfs3\_fh\_t \*fh\_extend(nfs\_fh3 fh, uint32 dev, uint64 ino, uint32 gen); 35](#_Toc366092626)

[4.6. post\_op\_fh3 fh\_extend\_post(nfs\_fh3 fh, uint32 dev, uint64 ino, uint32 gen); 35](#_Toc366092627)

[4.7. post\_op\_fh3 fh\_extend\_type(nfs\_fh3 fh, const char \*path, unsigned int type); 35](#_Toc366092628)

[4.8. int nfh\_valid(nfs\_fh3 fh); 35](#_Toc366092629)

[4.8.1. 使用说明 35](#_Toc366092630)

[4.8.2. 代码重构 35](#_Toc366092631)

[4.9. int fh\_valid(unfs3\_fh\_t fh); 35](#_Toc366092632)

[4.9.1. 使用说明 35](#_Toc366092633)

[4.9.2. 代码重构 35](#_Toc366092634)

[4.10. u\_int fh\_length(const unfs3\_fh\_t \*fh); 36](#_Toc366092635)

[4.10.1. 使用说明 36](#_Toc366092636)

[4.10.2. 代码重构 36](#_Toc366092637)

[5. fh\_cache.c 36](#_Toc366092638)

[5.1. void fh\_cache\_init(void) 36](#_Toc366092639)

[5.1.1. 使用说明 36](#_Toc366092640)

[5.1.2. 核心代码 37](#_Toc366092641)

[5.2. char \*fh\_decomp(nfs\_fh3 fh); 37](#_Toc366092642)

[5.3. fh\_comp(const char \*path, struct svc\_req \* rqstp, int need\_dir) 37](#_Toc366092643)

[5.3.1. 使用说明 37](#_Toc366092644)

[5.3.2. 核心代码 38](#_Toc366092645)

[5.3.3. 代码重构 38](#_Toc366092646)

[6. fd\_cache.c 38](#_Toc366092647)

[6.1. void fd\_cache\_init(void) 38](#_Toc366092648)

[6.1.1. 使用说明 38](#_Toc366092649)

[6.1.2. 核心代码 38](#_Toc366092650)

[6.2. void fd\_cache\_purge(void); 39](#_Toc366092651)

[6.3. void fd\_cache\_close\_inactive(void); 39](#_Toc366092652)

[7. user.c 39](#_Toc366092653)

[7.1. void get\_squash\_ids(void); 40](#_Toc366092654)

[8. attr.c 40](#_Toc366092655)

[8.1. post\_op\_attr get\_post\_cached(struct svc\_req \*req) 40](#_Toc366092656)

[8.2. pre\_op\_attr get\_post\_buf(backend\_statstruct buf, struct svc\_req \*req); 40](#_Toc366092657)

[8.3. post\_op\_attr get\_post\_stat(const char \*path, struct svc\_req \* req) 40](#_Toc366092658)

[8.4. static post\_op\_attr get\_post\_ll(const char \*path, uint32 dev, uint64 ino,struct svc\_req \*req) 41](#_Toc366092659)

[9. readdir.c 41](#_Toc366092660)

[9.1. READDIR3res read\_dir(const char \*path, cookie3 cookie, cookieverf3 verf, count3 count) 41](#_Toc366092661)

[safe.c 41](#_Toc366092662)

[Locate.c 41](#_Toc366092663)

[10. 系统调用 41](#_Toc366092664)

[lstat函数 41](#_Toc366092665)

# 主要流程

Mount流程

|  |
| --- |
| ================0 Request======================  ================1 Request======================  ================2 Request======================  ================3 Request======================  ================4 Request======================  Running daemon.c/nfs3\_program\_3()  NFSPROC3\_NULL  Running nfs.c----nfsproc3\_null\_3\_svc()  daemon.c/nfs3\_program\_3() --calling-- svc\_sendreply()  ================5 Request======================  ================6 Request======================  Running daemon.c/mountprog\_3()  MOUNTPROC\_NULL  Running mount.c/mountproc\_null\_3\_svc()  Result == NULL  daemon.c/mountprog\_3() --calling-- svc\_sendreply()  ================7 Request======================  Running daemon.c/mountprog\_3()  MOUNTPROC\_NULL  Running mount.c/mountproc\_null\_3\_svc()  Result == NULL  daemon.c/mountprog\_3() --calling-- svc\_sendreply()  ================8 Request======================  Running daemon.c/mountprog\_3()  MOUNTPROC\_MNT  Running mount.c/mountproc\_mnt\_3\_svc()  Running daemon.c/get\_remote()  Running daemon.c/get\_port()  mount.c/mountproc\_mnt\_3\_svc() --calling-- fh\_cache.c/fh\_comp()  dir:/tmp will be mounted.  Running fh\_cache.c----fh\_comp()  Running fh.c/fh\_comp\_raw()  Running daemon.c/get\_remote()  Running fh.c----get\_gen()  Running fh.c----fh\_valid()  Running fh\_cache.c----fh\_cache\_add()  Running fh\_cache.c----fh\_cache\_index()  Running fh\_cache.c----fh\_cache\_lru()  Running fh.c----fh\_valid()  Running mount.c----add\_mount()  Running daemon.c/get\_remote()  Running fh.c----fh\_length()  Result == ??  daemon.c/mountprog\_3() --calling-- svc\_sendreply()  ================9 Request======================  ================10 Request======================  Running daemon.c/nfs3\_program\_3()  NFSPROC3\_NULL  Running nfs.c----nfsproc3\_null\_3\_svc()  daemon.c/nfs3\_program\_3() --calling-- svc\_sendreply()  ================11 Request======================  ================12 Request======================  Running daemon.c/nfs3\_program\_3()  NFSPROC3\_FSINFO  Running nfs.c----nfsproc3\_fsinfo\_3\_svc  Running daemon.c/get\_socket\_type()  Running user.c----switch\_to\_root()  Running fh\_cache.c----fh\_decomp()  Running fh.c----nfh\_valid()  Running fh.c----fh\_length()  Running fh\_cache.c----fh\_cache\_lookup()  Running fh\_cache.c----fh\_cache\_index()  Running daemon.c/get\_remote()  Running user.c----switch\_user()  Running user.c----get\_gid()  Running user.c----mangle()  Running user.c----switch\_groups()  Running user.c----mangle()  Running user.c----get\_uid()  Running user.c----mangle()  Running attr.c----get\_post\_cached()  Running attr.c----get\_post\_buf()  daemon.c/nfs3\_program\_3() --calling-- svc\_sendreply()  ================13 Request======================  Running daemon.c/nfs3\_program\_3()  NFSPROC3\_PATHCONF  Running nfs.c----nfsproc3\_pathconf\_3\_svc  Running user.c----switch\_to\_root()  Running fh\_cache.c----fh\_decomp()  Running fh.c----nfh\_valid()  Running fh.c----fh\_length()  Running fh\_cache.c----fh\_cache\_lookup()  Running fh\_cache.c----fh\_cache\_index()  Running daemon.c/get\_remote()  Running user.c----switch\_user()  Running user.c----get\_gid()  Running user.c----mangle()  Running user.c----switch\_groups()  Running user.c----mangle()  Running user.c----get\_uid()  Running user.c----mangle()  Running attr.c----get\_post\_cached()  Running attr.c----get\_post\_buf()  daemon.c/nfs3\_program\_3() --calling-- svc\_sendreply()  ================14 Request======================  Running daemon.c/nfs3\_program\_3()  NFSPROC3\_FSINFO  Running nfs.c----nfsproc3\_fsinfo\_3\_svc  Running daemon.c/get\_socket\_type()  Running user.c----switch\_to\_root()  Running fh\_cache.c----fh\_decomp()  Running fh.c----nfh\_valid()  Running fh.c----fh\_length()  Running fh\_cache.c----fh\_cache\_lookup()  Running fh\_cache.c----fh\_cache\_index()  Running daemon.c/get\_remote()  Running user.c----switch\_user()  Running user.c----get\_gid()  Running user.c----mangle()  Running user.c----switch\_groups()  Running user.c----mangle()  Running user.c----get\_uid()  Running user.c----mangle()  Running attr.c----get\_post\_cached()  Running attr.c----get\_post\_buf()  daemon.c/nfs3\_program\_3() --calling-- svc\_sendreply()  ================15 Request======================  Running daemon.c/nfs3\_program\_3()  NFSPROC3\_ACCESS  Running nfs.c----nfsproc3\_access\_3\_svc  Running user.c----switch\_to\_root()  Running fh\_cache.c----fh\_decomp()  Running fh.c----nfh\_valid()  Running fh.c----fh\_length()  Running fh\_cache.c----fh\_cache\_lookup()  Running fh\_cache.c----fh\_cache\_index()  Running daemon.c/get\_remote()  Running user.c----switch\_user()  Running user.c----get\_gid()  Running user.c----mangle()  Running user.c----switch\_groups()  Running user.c----mangle()  Running user.c----get\_uid()  Running user.c----mangle()  Running attr.c----get\_post\_cached()  Running attr.c----get\_post\_buf()  Running user.c----is\_owner()  Running user.c----get\_uid()  Running user.c----mangle()  Running user.c----get\_uid()  Running user.c----mangle()  daemon.c/nfs3\_program\_3() --calling-- svc\_sendreply()  ================16 Request======================  Running daemon.c/nfs3\_program\_3()  NFSPROC3\_LOOKUP  Running nfs.c----nfsproc3\_lookup\_3\_svc  Running user.c----switch\_to\_root()  Running fh\_cache.c----fh\_decomp()  Running fh.c----nfh\_valid()  Running fh.c----fh\_length()  Running fh\_cache.c----fh\_cache\_lookup()  Running fh\_cache.c----fh\_cache\_index()  Running daemon.c/get\_remote()  Running user.c----switch\_user()  Running user.c----get\_gid()  Running user.c----mangle()  Running user.c----switch\_groups()  Running user.c----mangle()  Running user.c----get\_uid()  Running user.c----mangle()  Running nfs.c----cat\_name()  Running error.c----lookup\_err()  Running attr.c----get\_post\_stat()  Running attr.c----get\_post\_ll()  Running attr.c----get\_post\_buf()  daemon.c/nfs3\_program\_3() --calling-- svc\_sendreply()  ================17 Request======================  Running daemon.c/nfs3\_program\_3()  NFSPROC3\_LOOKUP  Running nfs.c----nfsproc3\_lookup\_3\_svc  Running user.c----switch\_to\_root()  Running fh\_cache.c----fh\_decomp()  Running fh.c----nfh\_valid()  Running fh.c----fh\_length()  Running fh\_cache.c----fh\_cache\_lookup()  Running fh\_cache.c----fh\_cache\_index()  Running daemon.c/get\_remote()  Running user.c----switch\_user()  Running user.c----get\_gid()  Running user.c----mangle()  Running user.c----switch\_groups()  Running user.c----mangle()  Running user.c----get\_uid()  Running user.c----mangle()  Running nfs.c----cat\_name()  Running error.c----lookup\_err()  Running attr.c----get\_post\_stat()  Running attr.c----get\_post\_ll()  Running attr.c----get\_post\_buf()  daemon.c/nfs3\_program\_3() --calling-- svc\_sendreply() |

# 主要的数据结构

## NFS的常数和文件模式

定义了六种基本常数来指明协议所用数组的大小。另外，象UNIX一样，NFS假定每个文件或目录一个指明其类型和存取保护的模式（mode）。列出了NFS模式整数的单个比特及其含义。定义直接对应于UNIX的stat函数的返回值。

|  |
| --- |
| nfs.h |
| #define NFS\_PORT 2049  #define NFS\_MAXDATA\_TCP 524288  #define NFS\_MAXDATA\_UDP 32768  #define NFS\_MAX\_UDP\_PACKET (NFS\_MAXDATA\_UDP + 4096) /\* The extra 4096 bytes are for the RPC header \*/  #define NFS\_MAXPATHLEN 1024  #define NFS\_MAXNAMLEN 255  #define NFS\_FIFO\_DEV -1  #define NFSMODE\_FMT 0170000  #define NFSMODE\_DIR 0040000 //这是个目录；类型是 NFDIR  #define NFSMODE\_CHR 0020000 //这是个字符专有文件；类型应该是NFCHR  #define NFSMODE\_BLK 0060000 //这是个块专有文件：类型应该是NFBLK  #define NFSMODE\_REG 0100000 //这是个普通文件；类型应该是NFREG  #define NFSMODE\_LNK 0120000 //这是个符号连接，类型应该是NFLNK  #define NFSMODE\_SOCK 0140000 //这是个有名的插口，类型应该是NFSOCK  #define NFSMODE\_FIFO 0010000 |

## NFS的调用返回值

协议定义一个常量枚举类型，被用于报告差错状态。每个远程调用都返回其中的一个值。22个NFS过程除了null过程外，都要返回一个状态，表明服务调用成功或者是出现错误。该协议的集合命名为stat，如下：

|  |
| --- |
| nfs.h |
| enum nfsstat3 {  NFS3\_OK = 0, //调用成功，成功的执行了服务过程  NFS3ERR\_PERM = 1, //调用者无权对文件对象进行操作  NFS3ERR\_NOENT = 2, //指定文件或目录不存在  NFS3ERR\_IO = 5, //I/O错误，在处理这个请求时硬件出现错误。  NFS3ERR\_NXIO = 6, //I/O错误，没有这个设备或者是地址  NFS3ERR\_ACCES = 13, //拒绝访问。调用者没有正确的权限来执行请求操作  NFS3ERR\_EXIST = 17, //指定文件已经存在  NFS3ERR\_XDEV = 18, //试图创建一个跨多个设备的硬链接  NFS3ERR\_NODEV = 19, //没有这样的设备  NFS3ERR\_NOTDIR = 20,//操作的对象不是目录  NFS3ERR\_ISDIR = 21, //操作对象是一个目录，而过程不知吃目录操作  NFS3ERR\_INVAL = 22, //参数无效  NFS3ERR\_FBIG = 27, //文件太大，超过了服务器的限制  NFS3ERR\_NOSPC = 28, //设备中没有空闲空间  NFS3ERR\_ROFS = 30, //试图在一个只读文件系统中执行修改操作  NFS3ERR\_MLINK = 31, //太多的硬链接  NFS3ERR\_NAMETOOLONG = 63, //文件名字太长  NFS3ERR\_NOTEMPTY = 66, //试图删除一个非空的目录  NFS3ERR\_DQUOT = 69, //用户在服务器上的资源超过了限制  NFS3ERR\_STALE = 70, //无效的文件句柄  NFS3ERR\_REMOTE = 71, //所指向的文件并不在服务器本地文件系统中  NFS3ERR\_BADHANDLE = 10001, //不合法的NFS文件句柄  NFS3ERR\_NOT\_SYNC = 10002, //在SETATTR操作时检测不匹配的同步更新  NFS3ERR\_BAD\_COOKIE = 10003, //READDIR，READDIRPLUS的cokkie过期  NFS3ERR\_NOTSUPP = 10004, //不支持此操作  NFS3ERR\_TOOSMALL = 10005, //缓冲或是请求太小  NFS3ERR\_SERVERFAULT = 10006,//一个不能确定的错误  NFS3ERR\_BADTYPE = 10007, //试图产生一个服务器不支持的类型的对象  NFS3ERR\_JUKEBOX = 10008, //服务器没能即使完成请求而客户重发请求  };  typedef enum nfsstat3 nfsstat3; |

## NFS的文件类型：

NFS使用和UNIX相同的基本文件类型。它定义了服务器在指定文件类型时可使用的枚举值。

|  |
| --- |
| nfs.h |
| enum ftype3 {  NF3REG = 1, //常规文件  NF3DIR = 2, //目录  NF3BLK = 3, //块设备文件  NF3CHR = 4, //特殊的设备文件  NF3LNK = 5, //符号链接  NF3SOCK = 6,//套接字  NF3FIFO = 7,//一个有名管道  };  typedef enum ftype3 ftype3; |

## NFS的文件句柄：

|  |
| --- |
| nfs.h |
| struct nfs\_fh3 {  struct {  u\_int data\_len;//表示file handle的大小  char \*data\_val;//存储file handle的内容。  } data;  };  typedef struct nfs\_fh3 nfs\_fh3; |

## NFS的文件属性结构：

NFS有一个获得关于某文件的信息的机制。在谈到文件信息时，NFS使用了术语file attribute。结构nfs\_fattr描述了NFS所提供的文件属性。

|  |
| --- |
| nfs.h |
| struct fattr3 {  ftype3 type; //文件的类型  mode3 mode; //保护模式bit  uint32 nlink; //一个文件硬链接的数目，即一个文件的不同名字的数目  uid3 uid; //用户ID  gid3 gid; //组ID  size3 size; //文件大小  size3 used; //文件实际占用的磁盘空间字节数  specdata3 rdev; //如果文件类型是NF3CHR或NF3BLK的设备文件  uint64 fsid; //文件系统标识符  fileid3 fileid; //在文件系统中的文件的独一无二的标志符  nfstime3 atime; //文件最后一次被访问的时间  nfstime3 mtime; //文件最后一次被修改的时间  nfstime3 ctime; //文件属性最后一次被修改的时间  };  typedef struct fattr3 fattr3; |
|  |

## Mount的常用返回值：

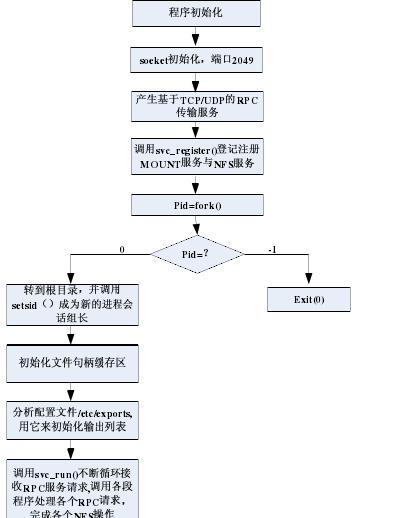
|  |
| --- |
| enum mountstat3 {  MNT3\_OK = 0, //无错误  MNT3ERR\_PERM = 1, //非文件所有者  MNT3ERR\_NOENT = 2, //文件或目录不存在  MNT3ERR\_IO = 5, //IO错误  MNT3ERR\_ACCES = 13, //没有访问权限  MNT3ERR\_NOTDIR = 20,//非目录  MNT3ERR\_INVAL = 22, //不合法请求  MNT3ERR\_NAMETOOLONG = 63, //文件名过长  MNT3ERR\_NOTSUPP = 10004, //操作不支持  MNT3ERR\_SERVERFAULT = 10006, //服务器错误  };  typedef enum mountstat3 mountstat3; |

# Daemon.c

后台处理程序，负责侦听端口2049，接受基于TCP/UDP的RPC数据包，并交与相应的程序过程处理各个请求包。

## main（）

### 流程



### 核心代码

|  |
| --- |
| … [parse\_options](#_parse_options(_)：解释命令行选项)(argc, argv); //程序选项  …  regenerate\_write\_verifier();  …  [register\_nfs\_service](#_register_nfs_service()：)(udptransp, tcptransp); //登记注册nfs服务  …  [register\_mount\_service](#_register_mount_service())(udptransp, tcptransp); //登记注册mount服务  …  create\_pid\_file();  [fh\_cache\_init();](#_void_fh_cache_init(void))  [fd\_cache\_init();](#_void_fh_cache_init(void)_1)  get\_squash\_ids();  exports\_parse(); //初始化输出列表  [unfs3\_svc\_run()](#_??unfs3_svc_run_(_);); //循环接受RPC请求 |

## parse\_options()

### 使用说明

解释命令行选项

|  |  |  |
| --- | --- | --- |
| -h | display this short option summary |  |
| -u | use unprivileged port for services |  |
| -d | do not detach from terminal |  |
| -e <file> | file to use instead of /etc/exports |  |
| -i <file> | write daemon pid to given file |  |
| -n <port> | port to use for NFS service |  |
| -m <port> | port to use for MOUNT service |  |
| -t | TCP only, do not listen on UDP ports |  |
| -p | do not register with the portmapper |  |
| -s | single user mode |  |
| -b | enable brute force file searching |  |
| -l <addr> | bind to interface with specified address |  |
| -r | report unreadable executables as readable |  |
| -T | test exports file and exit |  |

## register\_nfs\_service()：

### 使用说明

登记注册NFS服务。

### 核心代码

|  |
| --- |
| if (udptransp != NULL) {  /\* Register NFS service for UDP \*/  if (!svc\_register  (udptransp, NFS3\_PROGRAM, NFS\_V3, [nfs3\_program\_3](#_nfs3_program_3_()：NFS服务调度函数，由RPC发起。),  opt\_portmapper ? IPPROTO\_UDP : 0)) {  fprintf(stderr, "%s\n",  "unable to register (NFS3\_PROGRAM, NFS\_V3, udp).");  daemon\_exit(0);  }  }  if (tcptransp != NULL) {  /\* Register NFS service for TCP \*/  if (!svc\_register  (tcptransp, NFS3\_PROGRAM, NFS\_V3, [nfs3\_program\_3](#_nfs3_program_3_()：NFS服务调度函数，由RPC发起。),  opt\_portmapper ? IPPROTO\_TCP : 0)) {  fprintf(stderr, "%s\n",  "unable to register (NFS3\_PROGRAM, NFS\_V3, tcp).");  daemon\_exit(0);  } |

## nfs3\_program\_3 ()：

### 使用说明

NFS服务调度函数，由RPC发起

一共22个NFS过程，由[nfs.c](#_nfs.c)定义。

### 核心代码

一共22个NFS过程，由[nfs.c](#_nfs.c)定义

|  |  |
| --- | --- |
| nfsproc3\_null\_3\_svc() |  |
| nfsproc3\_getattr\_3\_svc() | //GETATTR。返回一个文件的属性：文件类型（一般文件，目录等）、访问权限、文件大小、文件的属主者及上一次访问时间等信息。 |
| nfsproc3\_setattr\_3\_svc() | //SETATTR。设置一个文件的属性。只允许设置文件属性的一个子集：访问权限、文件的属主、组的属主、文件大小、上次访问时间和上次修改时间。 |
|  | //LOOKUP。查找一个文件。 |
|  | //READ。从一个文件中读数据。客户说明文件的句柄、读操作的开始位置和读数据的最大字节数。 |
|  | //WRITE。对一个文件进行读写操作。客户说明文件的句柄、开始位置、写数据的字节数和要写的数据。 |
|  | //CREAT。创建一个文件。 |
|  | //REMOVE。删除一个文件。 |
|  | //RENAME。重命名一个文件。 |
|  | //LINK。为一个文件构造一个硬链接。硬链接是一个Unix的概念，指的是磁盘中的一个文件可以有任意多个目录项指向它。 |
|  | //SYMLINK。为一个文件创建一个符号链接。符号链接的一个包含另一个文件名字的文件。大多数引用符号链接的操作（例如，打开）实际上引用的是符号链接指向的文件。 |
|  | //READLINK。读一个符号链接。即返回符号链接所指的文件的名字。 |
|  | //MKDIR。创建一个目录。 |
|  | //RMDIR。删除一个目录。 |
|  | //READDIR。读一个目录。 |
|  | //ACCESS。检查问价访问权限。 |
|  | //MKNOD。创建一个Unix特殊文件。 |
|  | //READDIRPLUS。返回一个目录中的文件名字和它们的属性。 |
|  | //FSINFO。返回一个文件系统的静态信息。 |
|  | //FSSTAT。返回一个文件系统的动态信息。 |
|  | //PATHCONF。返回一个文件的POSIX.1信息。 |
|  | //COMMIT。将以前的异步写操作提交到外存中。 |

## register\_mount\_service()

### 使用说明

登记注册MOUNT服务。

### 核心代码

|  |
| --- |
| if (!svc\_register  (udptransp, MOUNTPROG, MOUNTVERS3, [mountprog\_3](#_mountprog_3():—》mount.c),  opt\_portmapper ? IPPROTO\_UDP : 0)) {  fprintf(stderr, "%s\n",  "unable to register (MOUNTPROG, MOUNTVERS3, udp).");  daemon\_exit(0);  }  if (!svc\_register  (tcptransp, MOUNTPROG, MOUNTVERS3, [mountprog\_3](#_mountprog_3():—》mount.c),  opt\_portmapper ? IPPROTO\_TCP : 0)) {  fprintf(stderr, "%s\n",  "unable to register (MOUNTPROG, MOUNTVERS3, tcp).");  daemon\_exit(0);  } |

## mountprog\_3()

### 使用说明

mount服务调度函数

一共7个mount过程，由[mount.c](#_nfs.c)定义。

### 核心代码

[**mount.c**](#_mount.c)**中的7个mount过程**

|  |
| --- |
| 1. MOUNTPROC\_NULL 0   //NULL。这是测试程序相应的ping过程，不执行任何操作。  extern void \* **mountproc\_null\_3\_svc**(void \*, struct svc\_req \*);   1. #define MOUNTPROC\_MNT 1   //MNT。挂载文件系统，NFS客户机传入需挂载的目录名，如果客户机有权访问这个目录，  // 则这个过程向NFS客户机返回目录的根NFS文件句柄。  extern mountres3 \* **mountproc\_mnt\_3\_svc**(dirpath \*, struct svc\_req \*);   1. #define MOUNTPROC\_DUMP 2   //DUMP。返回所有被挂载的文件系统的列表。  extern mountlist \* **mountproc\_dump\_3\_svc**(void \*, struct svc\_req \*);   1. #define MOUNTPROC\_UMNT 3   //UMNT。卸载文件系统，结束对服务器目录的挂载。  extern void \* **mountproc\_umnt\_3\_svc**(dirpath \*, struct svc\_req \*);   1. #define MOUNTPROC\_UMNTALL 4   //UMNTALL。卸载请求客户端的所有挂载项  extern void \* **mountproc\_umntall\_3\_svc(**void \*, struct svc\_req \*);   1. #define MOUNTPROC\_EXPORT 5   //EXPORT。列出该服务器所有可以挂载的文件系统。  extern exports \* **mountproc\_export\_3\_svc**(void \*, struct svc\_req \*); |

## unfs3\_svc\_run()

### 使用说明

循环接受 RPC 请求。它实际上是调用 select()监听之前创建的socket，并将收到的RPC调用分发给对应的RPC请求处理函数。

### 核心代码

|  |
| --- |
| …  for (;;) {  …  [fd\_cache\_close\_inactive](#_void_fd_cache_close_inactive(void);)(); //关闭没有使用的文件描述符  …  switch (select(FD\_SETSIZE, &readfds, NULL, NULL, &tv)) {  case -1: //select失败  …  return;  case 0: //select函数超时时返回0  …  continue;  default:  svc\_getreqset(&readfds);  }  }  … |

# mount.c

处理客户的挂载安装过程，正确处理返回给客户一个根文件句柄。

## 重构mountres3 \* mountproc\_mnt\_3\_svc(dirpath \*, struct svc\_req \*);

### 使用说明

1. 挂载文件系统，NFS客户机传入需挂载的目录名，如果客户机有权访问这个目录，则这个过程向NFS客户机返回目录的根NFS文件句柄。
2. 返回值result的数据结构mountres3：



### 核心代码

|  |
| --- |
| …  fh = fh\_comp(buf, rqstp, FH\_DIR);  //将buf中存储的路径转变为文件句柄内容，参见fh\_cache.c/ [fh\_comp()](#_fh_comp(const_char_*path,)  …  result.fhs\_status = MNT3\_OK;  result.mountres3\_u.mountinfo.fhandle.fhandle3\_len = fh\_length(&fh);  result.mountres3\_u.mountinfo.fhandle.fhandle3\_val = (char \*) &fh;  result.mountres3\_u.mountinfo.auth\_flavors.auth\_flavors\_len = 1;  result.mountres3\_u.mountinfo.auth\_flavors.auth\_flavors\_val = &auth; |

### 代码重构

重构fh\_cache.c/ [fh\_comp()](#_fh_comp(const_char_*path,)

## 重构void \* mountproc\_umnt\_3\_svc(dirpath \*, struct svc\_req \*);

### 使用说明

1. 卸载文件系统，结束对服务器目录的挂载。
2. 返回值为NULL

### 核心代码

|  |
| --- |
| …  remove\_mount(\*argp, rqstp); //从挂载列表删除  if (mount\_cnt == 0) fd\_cache\_purge(); //如果没有客户机链接，清空所有使用的文件描述符，参见fd\_cache.c/[fd\_cache\_purge()](#_void_fd_cache_purge(void););  … |

### 代码重构

待定

## void \* mountproc\_umntall\_3\_svc(void \*, struct svc\_req \*);

### 使用说明

卸载文件系统，结束对服务器目录的挂载。

返回值为NULL

### 核心代码

|  |
| --- |
| …  remove\_mount(NULL, rqstp); //从挂载列表删除  if (mount\_cnt == 0) fd\_cache\_purge(); //如果没有客户机链接，清空所有使用的文件描述符，参见fd\_cache.c/[fd\_cache\_purge()](#_void_fd_cache_purge(void););  … |

### 代码重构

待定

## void \* mountproc\_null\_3\_svc(void \*, struct svc\_req \*)

### 使用说明

这是测试程序相应的ping过程，不执行任何操作。

### 代码重构

不需重构

## exports \* mountproc\_export\_3\_svc(void \*, struct svc\_req \*);

### 使用说明

列出该服务器所有可以挂载的文件系统。

### 代码重构

不需重构

## mountlist \* mountproc\_dump\_3\_svc(void \*, struct svc\_req \*);

### 使用说明

返回所有被挂载的文件系统的列表

### 代码重构

不需重构

## static void add\_mount(const char \*path, struct svc\_req \*rqstp)

### 使用说明

增加entry到mount list

### 代码重构

不需重构

## static void remove\_mount(const char \*path, struct svc\_req \*rqstp)

### 使用说明

从mount list删除entry

### 代码重构

不需重构

# nfs.c

NFS 协议的 22 个协议过程的处理

## 不重构void \*nfsproc3\_null\_3\_svc(U(void \*argp), U(struct svc\_req \*rqstp))

### 使用说明

NFS\_NULL（过程0）：按照习惯，在任何RPC程序中过程0被称为空，因为它没有任何动作。应用程序可以调用它来测试某个服务器是否响应。

## 重构FSINFO3res \*nfsproc3\_fsinfo\_3\_svc(FSINFO3args \* argp, struct svc\_req \* rqstp)

### 使用说明

返回一个文件系统的静态信息。

### 核心代码

|  |
| --- |
| …  PREP(path, argp->fsroot);  result.FSINFO3res\_u.resok.obj\_attributes = get\_post\_cached(rqstp);  … |

### 重构内容

|  |
| --- |
| fh\_cache.c/[fh\_decomp();](#_char_*fh_decomp(nfs_fh3_fh);)  attr.c/[get\_post\_cached()](#_post_op_attr_get_post_cached(struct) |

## 重构PATHCONF3res \*nfsproc3\_pathconf\_3\_svc(PATHCONF3args \* argp, struct svc\_req \* rqstp)

### 使用说明

返回一个文件的POSIX.1信息。

### 核心代码

|  |
| --- |
| …  PREP(path, argp->fsroot);  result.PATHCONF3res\_u.resok.obj\_attributes = get\_post\_cached(rqstp);  … |

### 重构内容

|  |
| --- |
| fh\_cache.c/[fh\_decomp();](#_char_*fh_decomp(nfs_fh3_fh);)  attr.c/[get\_post\_cached()](#_post_op_attr_get_post_cached(struct) |

## 重构ACCESS3res \* nfsproc3\_access\_3\_svc(ACCESS3args \*, struct svc\_req \*);

### 使用说明

检查问价访问权限。

### 核心代码

|  |
| --- |
| …  PREP(path, argp->fsroot);  post = get\_post\_cached(rqstp);  … |

### 重构内容

|  |
| --- |
| fh\_cache.c/[fh\_decomp();](#_char_*fh_decomp(nfs_fh3_fh);)  attr.c/[get\_post\_cached()](#_post_op_attr_get_post_cached(struct) |

## 重构GETATTR3res \*nfsproc3\_getattr\_3\_svc(GETATTR3args \* argp, struct svc\_req \* rqstp)

### 使用说明

1. NFS\_GETATTR（过程1）：客户机调用过程1来得到某个文件的属性，包括保护模式、文件拥有者、大小以及最近存取时间等项。
2. 返回值result数据结构GETATTR3res：



1. 发送值Post的数据结构：



### 核心代码

|  |
| --- |
| …  char \*path;  post\_op\_attr post;  PREP(path, argp->object); //将文件句柄转为路径，主要调用fh\_cache.c/[fh\_decomp()](#_char_*fh_decomp(nfs_fh3_fh);)  post = get\_post\_cached(rqstp); //  result.status = NFS3\_OK;  result.GETATTR3res\_u.resok.obj\_attributes =post.post\_op\_attr\_u.attributes;  return &result;  … |

### 重构内容

|  |
| --- |
| fh\_cache.c/[fh\_decomp();](#_char_*fh_decomp(nfs_fh3_fh);)  attr.c/[get\_post\_cached()](#_post_op_attr_get_post_cached(struct) |

## 重构READDIR3res \* nfsproc3\_readdir\_3\_svc(READDIR3args \*, struct svc\_req \*);

### 使用说明

返回一个文件的POSIX.1信息。

### 核心代码

|  |
| --- |
| …  PREP(path, argp->dir);  result = read\_dir(path, argp->cookie, argp->cookieverf, argp->count);  result.READDIR3res\_u.resok.dir\_attributes = get\_post\_stat(path, rqstp);  … |

### 重构内容

|  |
| --- |
| fh\_cache.c/[fh\_decomp();](#_char_*fh_decomp(nfs_fh3_fh);)  readdir.c/[read\_dir()](#_READDIR3res_read_dir(const_char)  attr.c/[get\_post\_stat()](#_post_op_attr_get_post_stat(const_ch) |

## 重构READDIRPLUS3res \* nfsproc3\_readdirplus\_3\_svc(READDIRPLUS3args \*, struct svc\_req \*);

### 使用说明

返回一个目录中的文件名字和它们的属性。

### 重构内容

无

## READ3res \* nfsproc3\_read\_3\_svc(READ3args \*, struct svc\_req \*);

## READLINK3res \* nfsproc3\_readlink\_3\_svc(READLINK3args \*, struct svc\_req \*);

## WRITE3res \* nfsproc3\_write\_3\_svc(WRITE3args \*, struct svc\_req \*);

## CREATE3res \* nfsproc3\_create\_3\_svc(CREATE3args \*, struct svc\_req \*);

## MKDIR3res \* nfsproc3\_mkdir\_3\_svc(MKDIR3args \*, struct svc\_req \*);

## SYMLINK3res \* nfsproc3\_symlink\_3\_svc(SYMLINK3args \*, struct svc\_req \*);

## MKNOD3res \* nfsproc3\_mknod\_3\_svc(MKNOD3args \*, struct svc\_req \*);

## LOOKUP3res \* nfsproc3\_lookup\_3\_svc(LOOKUP3args \*, struct svc\_req \*);

## REMOVE3res \* nfsproc3\_remove\_3\_svc(REMOVE3args \*, struct svc\_req \*);

## RMDIR3res \* nfsproc3\_rmdir\_3\_svc(RMDIR3args \*, struct svc\_req \*);

## RENAME3res \* nfsproc3\_rename\_3\_svc(RENAME3args \*, struct svc\_req \*);

## LINK3res \* nfsproc3\_link\_3\_svc(LINK3args \*, struct svc\_req \*);

## FSSTAT3res \* nfsproc3\_fsstat\_3\_svc(FSSTAT3args \*, struct svc\_req \*);

## SETATTR3res \* nfsproc3\_setattr\_3\_svc(SETATTR3args \*, struct svc\_req \*);

## COMMIT3res \* nfsproc3\_commit\_3\_svc(COMMIT3args \*, struct svc\_req \*);

## int nfs3\_program\_3\_freeresult (SVCXPRT \*, xdrproc\_t, caddr\_t);

# fh.c

实现文件句柄的相关操作，主要实现的函数：

fh\_valid( )：验证句柄是否有效。

nfh\_valid( )：验证nfs句柄是否有效。

fh\_length()：计算文件句柄的真实长度。

fh\_decomp\_raw()：解析文件句柄，由文件句柄得到文件路径名。

fh\_comp\_raw( )：构造文件句柄。

fh\_rec()：

get\_gen()：

fh\_extend()：

fh\_extend\_post()：

fh\_extend\_type()：

## 重构 unfs3\_fh\_t fh\_comp\_raw(const char \*path, struct svc\_req \*rqstp, int need\_dir);

### 使用说明

compose a filehandle for a given path

path: path to compose fh for；文件路径名

rqstp: If not NULL, generate special FHs for removables

need\_dir: if not 0, path must point to a directory

返回值fh数据结构unfs3\_fh\_t：

|  |
| --- |
| typedef struct {  uint32 dev;  uint64 ino;  uint32 gen;  uint32 pwhash;  unsigned char len;  unsigned char inos[FH\_MAXLEN];  } \_\_attribute\_\_((packed)) unfs3\_fh\_t; |

### 原核心代码结构

|  |
| --- |
| char work[NFS\_MAXPATHLEN];  unfs3\_fh\_t fh;  backend\_statstruct buf; //文件属性  int res;  char \*last;  int pos = 0;  fh.len = 0;  …  res = [backend\_lstat](#_lstat函数_1)(path, &buf);//获取文件信息  if (res == -1)  return invalid\_fh;  /\* check for dir if need\_dir is set \*/  if (need\_dir != 0 && !S\_ISDIR(buf.st\_mode))  return invalid\_fh;  fh.dev = buf.st\_dev;  fh.ino = buf.st\_ino;  // obtain inode generation number if possible  fh.gen = [backend\_get\_gen](#_uint32_get_gen(backend_statstruct_o)(buf, FD\_NONE, path);  /\* special case for root directory \*/  if (strcmp(path, "/") == 0)  return fh;  strcpy(work, path);  last = work;  do {  \*last = '/';  last = strchr(last + 1, ‘/');  if (last != NULL)  \*last = 0;  res = backend\_lstat(work, &buf);  if (res == -1) {  return invalid\_fh;  }  /\* store 8 bit hash of the component's inode \*/  fh.inos[pos] = FH\_HASH(buf.st\_ino);  pos++;  } while (last && pos < FH\_MAXLEN);  if (last) /\* path too deep for filehandle \*/  return invalid\_fh;  fh.len = pos;  return fh; |

## static int fh\_rec(const unfs3\_fh\_t \* fh, int pos, const char \*lead,char \*result)

### 使用说明

递归搜索

\* fh: filehandle being resolved

\* pos: position in filehandles path inode array

\* lead: current directory for search

\* result: where to store path if seach is complete

返回值

False/True

### 核心代码结构

## uint32 get\_gen(backend\_statstruct obuf, int fd, const char \*path);

### 使用说明

obtain inode generation number if possible

obuf: filled out stat buffer (must be given!)

fd: open fd to file or FD\_NONE (-1) if no fd open；文件描述符

path: path to object in case we need to open it here；文件路径名

失败时返回0。

## unfs3\_fh\_t \*fh\_extend(nfs\_fh3 fh, uint32 dev, uint64 ino, uint32 gen);

## post\_op\_fh3 fh\_extend\_post(nfs\_fh3 fh, uint32 dev, uint64 ino, uint32 gen);

## post\_op\_fh3 fh\_extend\_type(nfs\_fh3 fh, const char \*path, unsigned int type);

## char \*fh\_decomp\_raw(const unfs3\_fh\_t \*fh);

### 使用说明

resolve a filehandle into a path

不能找到对象时，返回NULL

### 核心代码结构

|  |
| --- |
| …  rec = fh\_rec(fh, 0, "/", result);//参见fh.c/[fh\_rec()](#_static_int_fh_rec(const)  if (rec) return result;  … |

### 代码重构

重构fh.c/[fh\_rec()](#_static_int_fh_rec(const)

## int nfh\_valid(nfs\_fh3 fh);

### 使用说明

检查nfs文件句柄是否正确

### 代码重构

不需重构

## int fh\_valid(unfs3\_fh\_t fh);

### 使用说明

检查文件句柄是否正确

### 代码重构

不需重构

## u\_int fh\_length(const unfs3\_fh\_t \*fh);

### 使用说明

计算文件句柄的真实长度

### 代码重构

不需重构

# fh\_cache.c

数据结构static unfs3\_cache\_t

|  |
| --- |
| typedef struct {  int fd; /\* open file descriptor \*/  int kind; /\* read or write \*/  time\_t use; /\* last use \*/  uint32 dev; /\* device \*/  uint64 ino; /\* inode \*/  uint32 gen; /\* inode generation \*/  } fd\_cache\_t;  static fd\_cache\_t fd\_cache[FD\_ENTRIES]; |
| #define FD\_ENTRIES 256 |

## void fh\_cache\_init(void)

### 使用说明

初始化fh\_chche

数据结构

|  |
| --- |
| typedef struct {  uint32 dev; /\* device \*/  uint64 ino; /\* inode \*/  char path[NFS\_MAXPATHLEN]; /\* pathname \*/  unsigned int use; /\* last use \*/  } unfs3\_cache\_t;  static unfs3\_cache\_t fh\_cache[CACHE\_ENTRIES]; |

### 核心代码

|  |
| --- |
| …  memset(fh\_cache, 0, sizeof(unfs3\_cache\_t) \* CACHE\_ENTRIES);  … |

数据结构static unfs3\_cache\_t

|  |
| --- |
| typedef struct {  uint32 dev; /\* device \*/  uint64 ino; /\* inode \*/  char path[NFS\_MAXPATHLEN]; /\* pathname \*/  unsigned int use; /\* last use \*/  } unfs3\_cache\_t;  static unfs3\_cache\_t fh\_cache[CACHE\_ENTRIES]; |
| #define CACHE\_ENTRIES 4096 |

## char \*fh\_decomp(nfs\_fh3 fh);

从根目录开始，递归遍历各文件的属性，核对fh->dev与fh->ino查找到fh对应的文件，最后返回path

## fh\_comp(const char \*path, struct svc\_req \* rqstp, int need\_dir)

### 使用说明

将路径转变为文件句柄

返回值res数据结构unfs3\_fh\_t：

|  |
| --- |
| typedef struct {  uint32 dev;  uint64 ino;  uint32 gen;  uint32 pwhash;  unsigned char len;  unsigned char inos[FH\_MAXLEN];  } \_\_attribute\_\_((packed)) unfs3\_fh\_t; |

### 核心代码

|  |
| --- |
| …  res = fh\_comp\_raw(path, rqstp, need\_dir);//参见fh.c/ [fh\_comp\_raw()](#_fh_comp_raw(const_char_*path,)  if (fh\_valid(res))  /\* add to cache for later use \*/  fh\_cache\_add(res.dev, res.ino, path);  res.pwhash = export\_password\_hash;  … |

### 代码重构

重构fh.c/ [fh\_comp\_raw()](#_fh_comp_raw(const_char_*path,)

unfs3\_fh\_t \*fh\_comp\_ptr(const char \*path, struct svc\_req \*rqstp, int need\_dir);

char \*fh\_cache\_add(uint32 dev, uint64 ino, const char \*path);

# fd\_cache.c

## void fd\_cache\_init(void)

### 使用说明

初始化fd\_chche

数据结构

|  |
| --- |
| typedef struct {  uint32 dev; /\* device \*/  uint64 ino; /\* inode \*/  char path[NFS\_MAXPATHLEN]; /\* pathname \*/  unsigned int use; /\* last use \*/  } unfs3\_cache\_t;  static unfs3\_cache\_t fh\_cache[CACHE\_ENTRIES]; |
| #define CACHE\_ENTRIES 4096 |

### 核心代码

|  |
| --- |
| …  for (i = 0; i < FD\_ENTRIES; i++) {  fd\_cache[i].fd = -1;  fd\_cache[i].kind = UNFS3\_FD\_READ;  fd\_cache[i].use = 0;  fd\_cache[i].dev = 0;  fd\_cache[i].ino = 0;  fd\_cache[i].gen = 0;  }  … |

int fd\_open(const char \*path, nfs\_fh3 fh, int kind, int allow\_caching);

int fd\_close(int fd, int kind, int really\_close);

int fd\_sync(nfs\_fh3 nfh);

## void fd\_cache\_purge(void);

## void fd\_cache\_close\_inactive(void);

# user.c

处理权限相关操作

int get\_uid(struct svc\_req \*req);

int mangle\_uid(int id);

int mangle\_gid(int id);

int is\_owner(int owner, struct svc\_req \*req);

int has\_group(int group, struct svc\_req \*req);

## void switch\_to\_root();

### 使用说明

转为root用户权限

### 核心代码

|  |
| --- |
| …  backend\_setegid(0);  backend\_seteuid(0);  … |

### 代码重构

不需重构

void switch\_user(struct svc\_req \*req);

void read\_executable(struct svc\_req \*req, backend\_statstruct buf);

void read\_by\_owner(struct svc\_req \*req, backend\_statstruct buf);

void write\_by\_owner(struct svc\_req \*req, backend\_statstruct buf);

## void get\_squash\_ids(void);

# attr.c

处理属性相关操作，主要实现函数：

## post\_op\_attr get\_post\_cached(struct svc\_req \*req)

|  |
| --- |
| …  return [get\_post\_buf(st\_cache, req);](#_pre_op_attr_get_post_buf(backend_st)  … |

## pre\_op\_attr get\_post\_buf(backend\_statstruct buf, struct svc\_req \*req);

## post\_op\_attr get\_post\_stat(const char \*path, struct svc\_req \* req)

|  |
| --- |
| …  return [get\_post\_ll](#_static_post_op_attr_get_post_ll(con)(path, st\_cache.st\_dev, st\_cache.st\_ino, req);  … |

## static post\_op\_attr get\_post\_ll(const char \*path, uint32 dev, uint64 ino,struct svc\_req \*req)

# readdir.c

## READDIR3res read\_dir(const char \*path, cookie3 cookie, cookieverf3 verf, count3 count)

# safe.c

实现文件对象的完整性检测相关操作，主要函数：

md5\_process( )：利用 MD5 算法对文件数据签名。

digest\_add( )：保存一文件与其对应的数字签名。

digest\_equal( )：比较两数字签名是否一致。

# Locate.c

char \*locate\_file(uint32 dev, uint64 ino);

# 系统调用

## lstat函数

|  |  |
| --- | --- |
| 函数名 | lstat |
| 功能 | 获取一些文件相关信息 |
| 用法 | int lstat(const char \*path, struct stat \*buf); |
| 参数 | path：文件路径名。  filedes：文件描述词。  buf：是以下结构体的指针  struct stat {  dev\_t st\_dev; /\* 文件所在设备的标识 \*/  ino\_t st\_ino; /\* 文件结点号 \*/  mode\_t st\_mode; /\* 文件保护模式 \*/  nlink\_t st\_nlink; /\* 硬连接数 \*/  uid\_t st\_uid; /\* 文件用户标识 \*/  gid\_t st\_gid; /\* 文件用户组标识 \*/  dev\_t st\_rdev; /\* 文件所表示的特殊设备文件的设备标识 \*/  off\_t st\_size; /\* 总大小，字节为单位 \*/  blksize\_t st\_blksize; /\* 文件系统的块大小 \*/  blkcnt\_t st\_blocks; /\* 分配给文件的块的数量，512字节为单元 \*/  time\_t st\_atime; /\* 最后访问时间 \*/  time\_t st\_mtime; /\* 最后修改时间 \*/  time\_t st\_ctime; /\* 最后状态改变时间 \*/  }; |
| 返回说明 | 成功执行时，返回0。失败返回-1，errno被设为以下的某个值  EBADF： 文件描述词无效  EFAULT： 地址空间不可访问  ELOOP： 遍历路径时遇到太多的符号连接  ENAMETOOLONG：文件路径名太长  ENOENT：路径名的部分组件不存在，或路径名是空字串  ENOMEM：内存不足  ENOTDIR：路径名的部分组件不是目录 |