File System

OS Study Session #8

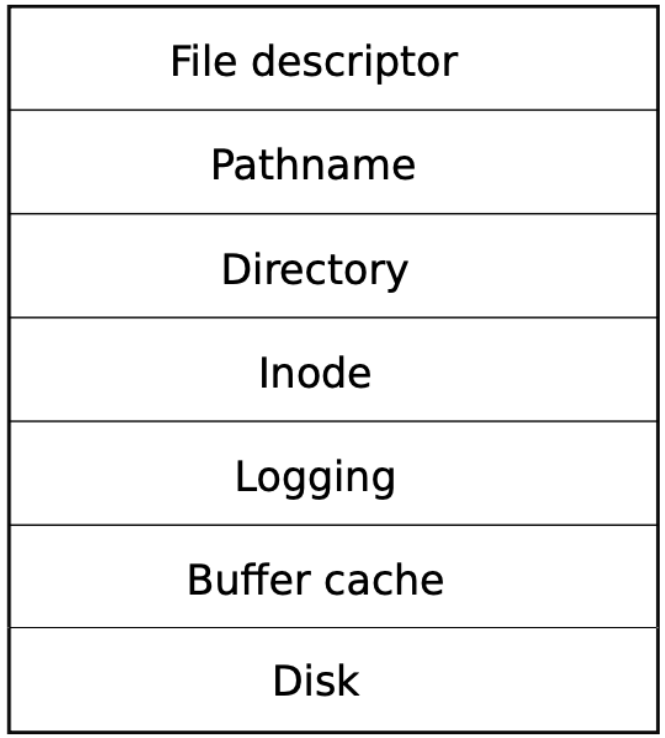


Figure 8.1: Layers of the xv6 file system.

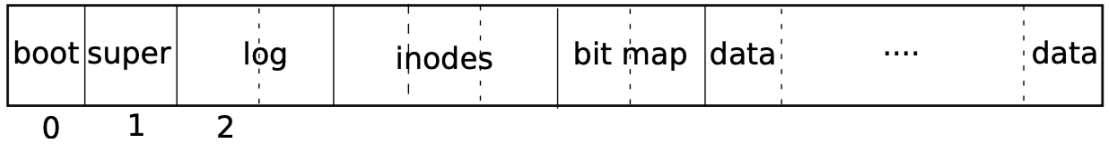


Figure 8.2: Structure of the xv6 file system.

Large files

(moderate)

Problem

- Increase the maximum size of an xv6 file
- Original xv6 file implementation :
  - 12 *direct* block numbers
  - one *singly-indirect* block number (refers to a block that holds up to 256 more blocks)
  - Total of 12+256 = **268 blocks**
- Support a ***doubly-indirect block*** in each inode

Block Pointers

Direct block

Holds 11 addresses

```
ip->addrs[0]  -->  Data Block 0
ip->addrs[1]  -->  Data Block 1
...
ip->addrs[10] -->  Data Block 10
```

Singly-Indirect block

Holds 256 addresses

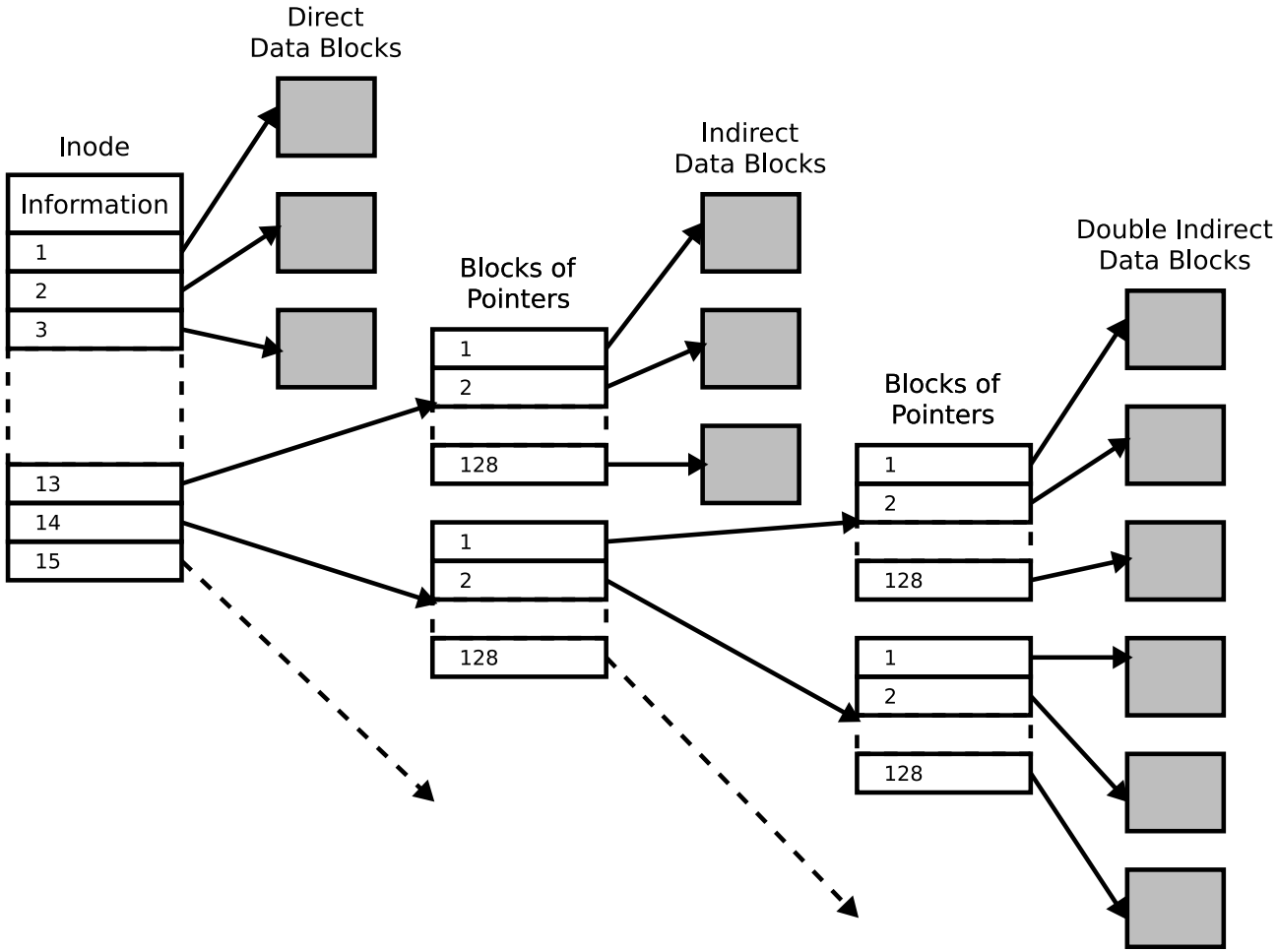
```
ip->addrs[11]  -->  Indirect Block  (holds 256 addresses)
                      |
                      +--> Data Block 11
                      +--> Data Block 12
                      ...
                      +--> Data Block 266
```

Doubly-Indirect block

Holds 256 * 256 addresses = 65,536

```
ip->addrs[12]  -->  Doubly-Indirect Block (256 pointers)
                      |
                      +--> Indirect Block 0 (256 pointers)
                      |       |
                      |       +--> Data Block 267
                      |       +--> ...
                      |
                      +--> Indirect Block 1 (256 pointers)
                      |       |
                      |       +--> Data Block ...
                      |
                      ...
                      +--> Indirect Block 255
                              |
                              +--> Data Block 65802
```

Block Pointers



Change inode structure on fs.h

```
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define MAXFILE (NDIRECT + NINDIRECT + NINDIRECT * NINDIRECT)

// On-disk inode structure
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEVICE only)
  short minor;          // Minor device number (T_DEVICE only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+2];   // Data block addresses: 11 direct + 1 single indirect + 1 double indirect
};
```

Implement bmap()

```
static uint
bmap(struct inode *ip, uint bn)
{
  uint addr, *a;
  struct buf *bp;

  if(bn < NDIRECT){
    // direct blocks
    if((addr = ip->addrs[bn]) == 0){
      addr = balloc(ip->dev);
      if(addr == 0)
        return 0;
      ip->addrs[bn] = addr;
    }
    return addr;
  }

  bn -= NDIRECT;
```

```
  if(bn < NINDIRECT){
    // load singly-indirect
    if((addr = ip->addrs[NDIRECT]) == 0){
      addr = balloc(ip->dev);
      if(addr == 0)
        return 0;
      ip->addrs[NDIRECT] = addr;
    }
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn]) == 0){
      addr = balloc(ip->dev);
      if(addr){
        a[bn] = addr;
        log_write(bp);
      }
    }
    brelse(bp);
    return addr;
  }
```

```
  bn -= NINDIRECT;

  if(bn < NINDIRECT * NINDIRECT){
    // load doubly-indirect
    if((addr = ip->addrs[NDIRECT + 1]) == 0){
      addr = balloc(ip->dev);
      if(addr == 0)
        return 0;
      ip->addrs[NDIRECT + 1] = addr;
    }

    // read doubly-indirect block
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;

    uint i1 = bn / NINDIRECT;
    uint i2 = bn % NINDIRECT;

    if((addr = a[i1]) == 0){
      addr = balloc(ip->dev);
      if(addr == 0){
        brelse(bp);
        return 0;
      }
      a[i1] = addr;
      log_write(bp);
    }
    brelse(bp);

    // read singly-indirect block pointed to by the doubly-indirect
    bp = bread(ip->dev, a[i1]);
    a = (uint*)bp->data;
    if((addr = a[i2]) == 0){
      addr = balloc(ip->dev);
      if(addr){
        a[i2] = addr;
        log_write(bp);
      }
    }
    brelse(bp);
    return addr;
  }
```

- If bn < 11: direct blocks
- If bn < 11 + 256: singly-indirect
- If bn < 11 + 256 + 256×256: doubly-indirect

Implement itrunc()

```
void
itrunc(struct inode *ip)
{
  int i, j;
  struct buf *bp;
  uint *a;

  for(i = 0; i < NDIRECT; i++){
    if(ip->addrs[i]){
      bfree(ip->dev, ip->addrs[i]);
      ip->addrs[i] = 0;
    }
  }

  // free singly indirect block
  if(ip->addrs[NDIRECT]){
    bp = bread(ip->dev, ip->addrs[NDIRECT]);
    a = (uint*)bp->data;
    for(j = 0; j < NINDIRECT; j++){
      if(a[j])
        bfree(ip->dev, a[j]);
    }
    brelse(bp);
    bfree(ip->dev, ip->addrs[NDIRECT]);
    ip->addrs[NDIRECT] = 0;
  }
```

```
  // free doubly indirect blocks
  if(ip->addrs[NDIRECT + 1]){
    struct buf *bp1 = bread(ip->dev, ip->addrs[NDIRECT + 1]);
    uint *a1 = (uint*)bp1->data;

    for(i = 0; i < NINDIRECT; i++){
      if(a1[i]){
        struct buf *bp2 = bread(ip->dev, a1[i]);
        uint *a2 = (uint*)bp2->data;

        for(j = 0; j < NINDIRECT; j++){
          if(a2[j]){
            bfree(ip->dev, a2[j]);  // free data block
          }
        }

        brelse(bp2);
        bfree(ip->dev, a1[i]);  // free singly-indirect block
      }
    }

    brelse(bp1);
    bfree(ip->dev, ip->addrs[NDIRECT + 1]);  // free doubly-indirect block
    ip->addrs[NDIRECT + 1] = 0;
  }

  // reset filesize and update inode
  ip->size = 0;
  iupdate(ip);
}
```

Change structure on file.h

```
// in-memory copy of an inode
struct inode {
  uint dev;           // Device number
  uint inum;          // Inode number
  int ref;            // Reference count
  struct sleeplock lock; // protects everything below here
  int valid;          // inode has been read from disk?

  short type;         // copy of disk inode
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+2];
};
```

How the inodes are managed

Directory – Path – File Descriptor

Filesystem Layers

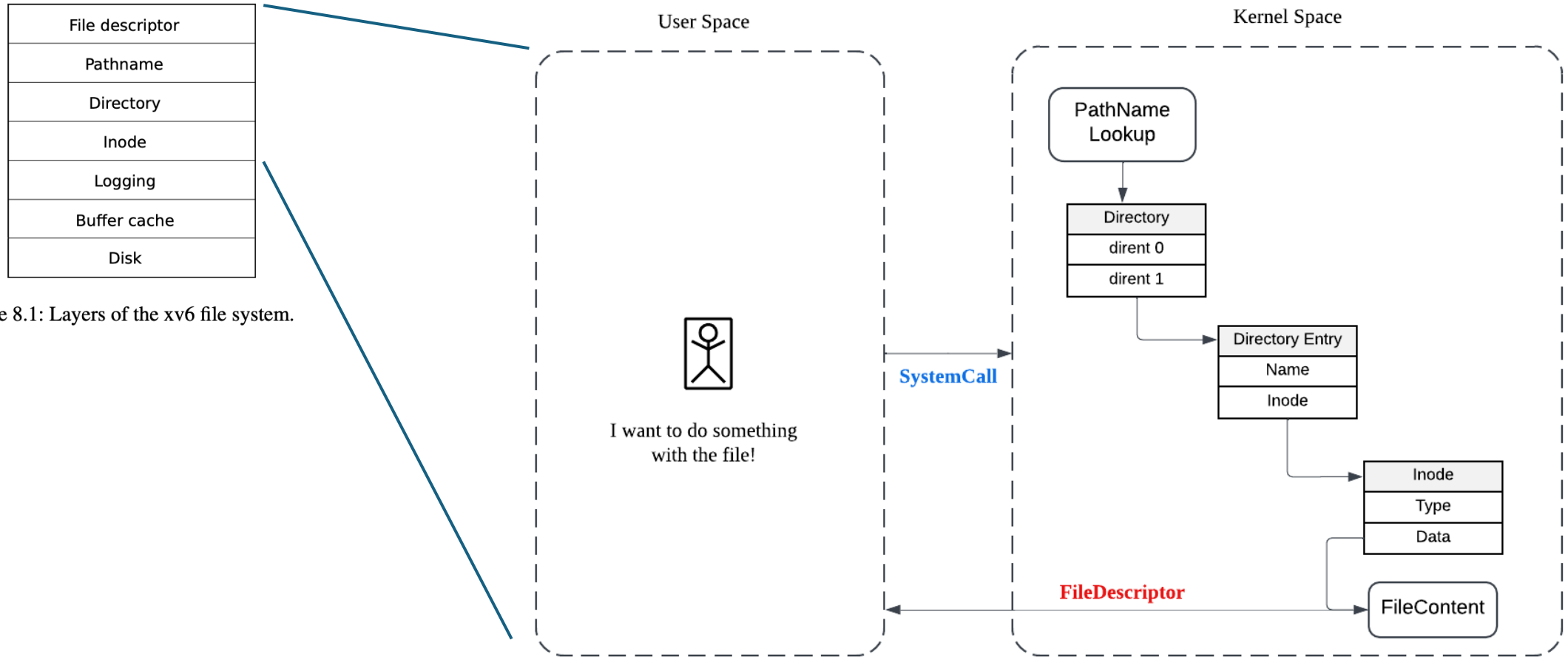


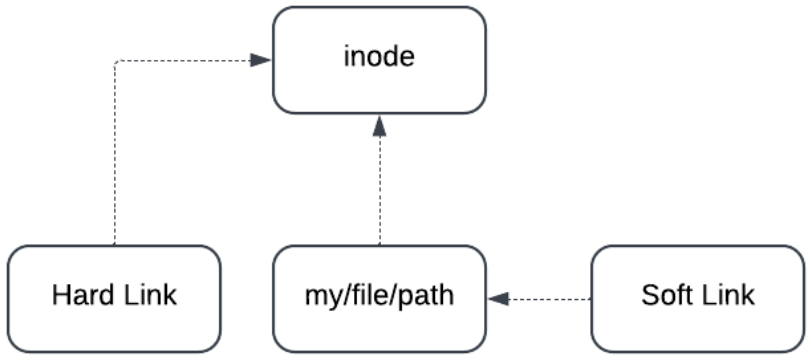
Figure 8.1: Layers of the xv6 file system.

Symbolic links

(moderate)

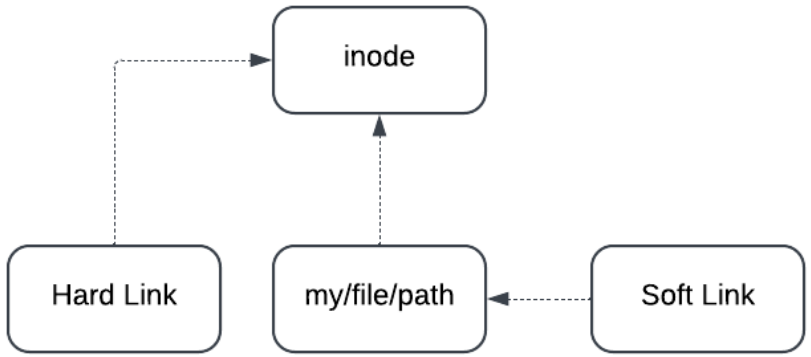
Problem

- Implementing a new system call: symlink(char *target, char*path)
- Two types of Link
  - Hard Link: Reference to the **actual inode**
  - Symbolic Link: Reference to the **name** (whatever the inode is)



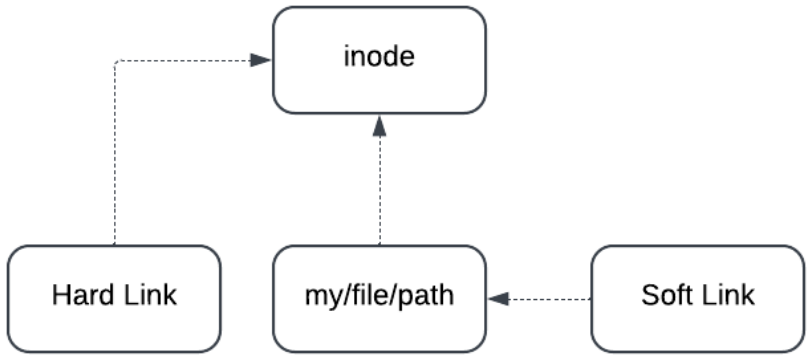
Hints

- Preparation for new system call
  - Create a new syscall number
  - Define a new syscall entry
- Implement ***sys_symlink(void)*** in kernel/sysfile.c
  - Define
    - Filetype: T_SYMLINK
    - File Open Flag: O_NOFOLLOW
  - No need to care if target inode exists
  - Save target path in inode's data block



Good reference: *sys_link*

- Limitation of the hard link
  - Target inode should be existing
  - Target inode should share the same device
- If above two conditions are met -> create directory entry with the target inode!



Good reference: *sys_link* (code)

```
// Create the path new as a link to the same inode as old.
uint64
sys_link(void)
{
  char name[DIRSIZ], new[MAXPATH], old[MAXPATH];
  struct inode *dp, *ip;

  if(argstr(0, old, MAXPATH) < 0 || argstr(1, new, MAXPATH) < 0)
    return -1;

  begin_op();
  if((ip = namei(old)) == 0){ // check if old (inode) exists
    end_op();
    return -1;
  }

  // INVARIANT: old does exist
  ilock(ip);
  if(ip->type == T_DIR){
    iunlockput(ip); // if it's a directory, we can't link it
    end_op();
    return -1;
  }

  ip->nlink++; // increment the link count of the inode
  iupdate(ip);
  iunlock(ip);

  if((dp = nameiparent(new, name)) == 0) // parent directory of the new must exist
    goto bad;
  ilock(dp);
  if(dp->dev != ip->dev || dirlink(dp, name, ip->inum) < 0){
    // new's parent directory must be on the same device as the existing node
    // and the new directory entry must be created in the parent directory
    iunlockput(dp);
    goto bad;
  }
  iunlockput(dp);
  iput(ip);

  end_op();

  return 0;

bad:
```

Implement system call ***symlink*** – (1)

- Some of predefinition for system call / new file type

user/usys.pl

```
27  27  entry("open");
28  28  entry("mknod");
29  29  entry("unlink");
30  30  entry("fstat");
31  31  entry("link");
32  32  entry("mkdir");
33  33  entry("chdir");
34  34  entry("dup");
35  35  entry("getpid");
36  36  entry("sbrk");
37  37  entry("sleep");
38  38  entry("uptime");
    39+ entry("symlink");
39  40
```

user/user.h

```
C user.h > write(int, const void *, int)
12  int exec(const char*, char**);
13  int open(const char*, int);
14  int mknod(const char*, short, short);
15  int unlink(const char*);
16  int fstat(int fd, struct stat*);
17  int link(const char*, const char*);
18  int mkdir(const char*);
19  int chdir(const char*);
20  int dup(int);
21  int getpid(void);
22  char* sbrk(int);
23  int sleep(int);
24  int uptime(void);
25+ int symlink(const char *target, const char *path
26
```

kernel/stat.h

```
C stat.h > ...
1  #define T_DIR     1   // Directory
2  #define T_FILE    2   // File
3  #define T_DEVICE  3   // Device
4+ #define T_SYMLINK 4   // Symbolic link
5
```

kernel/fcntl.h

```
fcntl.h > ...
1  #define O_RDONLY  0x000
2  #define O_WRONLY  0x001
3  #define O_RDWR    0x002
4  #define O_CREATE  0x200
5  #define O_TRUNC   0x400
6+ #define O_NOFOLLOW 0x800
```

Implement system call ***symlink*** – (2)

- Define a new function **sys_symlink(void)**
  - *dirlink*(struct inode *dp, char *name, uint inum)
  - *dirlookup*(struct inode *dp, char *name, uint *poff)

- Modify **sys_open(void)** for new file type (T_SYMLINK)
  - *writei*(struct inode *ip, ..., uint n)
  - *readi*(struct inode *ip, ..., uint n)
  - *namex*(char *path, int nameiparent, char *name)

Implement system call *symlink* – (3)

- Define a new function **sys_symlink(void)**
  - *dirlink*(struct inode *dp, char *name, uint inum)
  - *dirlookup*(struct inode *dp, char *name, uint *poff)

```
// Create the path new as a soft link to the target path name
uint64
sys_symlink(void)
{
  char new[MAXPATH], target[MAXPATH];
  struct inode *tp;

  if(argstr(0, target, MAXPATH) < 0 || argstr(1, new, MAXPATH) < 0)
    return -1;

  begin_op();
  if((tp = namei(new)) != 0){ // check if the inode with the same name as new exists
    end_op();
    return -1;
  }
  // make a new soft link with name new
  if((tp = create(new, T_SYMLINK, 0, 0)) == 0){ // create a new inode for the symlink
    end_op();
    return -1;
  }

  if (writei(tp, 0, (uint64)target, 0, MAXPATH) < 0){ // write a target path to the inode
    iunlockput(tp);
    end_op();
    return -1;
  }
```

Implement system call ***symlink*** – (2)

- Modify **sys_open(void)** for new file type (T_SYMLINK)
  - *writei*(struct inode *ip, ..., uint n)
  - *readi*(struct inode *ip, ..., uint n)
  - *namex*(char *path, int nameiparent, char *name)

```
}

if (ip->type == T_SYMLINK && (omode & O_NOFOLLOW) == 0) { // Follow the link until we reach
  if ((tp = looksimlink(ip)) == -1) {
    iunlockput(ip);
    end_op();
    return -1;
  }
  ilock(tp);
  ip = tp;
}

  return -1;
```

```
struct inode*
looksimlink(struct inode *ip) {
  int i;

  for (i = 0; i < 10; i++) {
    char target[MAXPATH];

    if ((readi(ip, 0, (uint64)target, 0, MAXPATH)) < 0) {
      return -1; // read the target path from the inode
    }
    if ((ip = namei(target)) == 0) {
      return -1;
    }
    if (ip->type != T_SYMLINK) { // if it's an actual file, return it
      return ip;
    }
  }
  iunlockput(ip);
  return -1; // if we reach here, it means we have a cyclic symlink
}
```