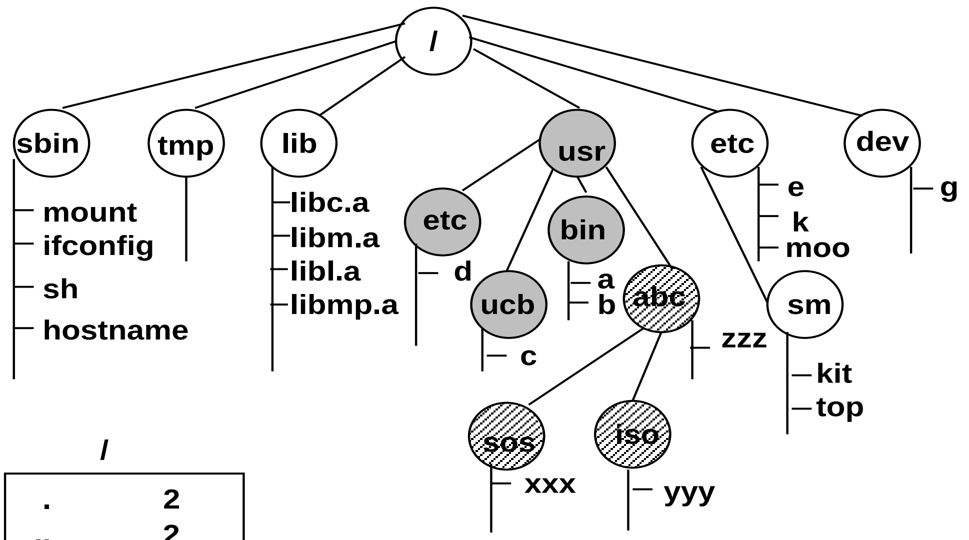
UNIX Directory Organization

UNIX directories are simple (generally ASCII) files which maintain a name to **inode** mapping which is convenient for people to use. All file objects are represented by one or more names located in a directory and pointing to the controlling inode for the named object. Each controlling inode keeps a *link count* which is incremented each time a new name is placed in a directory to point to the inode, and decremented each time a name is removed from a directory (by an *rm* command). When the link count on an inode reaches zero, the node is returned to the free I-list and its data blocks (if any exist) are returned to the free data block list. If you are interested in the inode which corresponds to a filename, use the **-i** option to the **ls** command:

```
%ls -lai

74116 drwxr-xr-x  2 bill       1024 Oct 17 21:49 .
38912 drwx--x--x 26 bill      54784 Nov 14 18:39 ..
74118 -rw-r--r--  1 bill       2714 May 10  1996 client.c
73822 -rw-r--r--  1 bill       3176 Aug 16 12:35 client_bsd.c
74149 -rw-r--r--  2 bill--|    2935 May 12  1996 client_sys5.c
74173 -rw-r--r--  1 bill  |    2775 May 10  1996 clientbsd.c
74021 -rw-r--r--  2 bill-||    3506 Aug 18 17:01 nc_test_bsd.c
74021 -rw-r--r--  2 bill_||    3506 Aug 18 17:01 nc_test_bsd_sun.c
73986 -rw-r--r--  1 bill  |    3494 Aug 18 14:36 new_test_client.c
74149 -rw-r--r--  2 bill__|    2935 May 12  1996 newclient.c
74148 -rw-r--r--  1 bill       3961 May 12  1996 server.c
```

UNIX Directory Organization



/

.	2
..	2
sbin	5
etc	12
usr	63
etc.....	

/etc

.	12
..	2
e	76
k	143
moo	76
sm	55
etc.....	

/usr

.	2
..	2
etc	12
ucb	44
bin	165
abc	432
etc.....	

/etc/sm

.	55
..	12
kit	76
top	1033
etc.....	

/usr/abc

.	2
..	2
sos	44
iso	63
zzz	721
etc.....	

Basic IO

The basic IO system calls include:

```
Create a file
SYNOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>

       int  creat (path, mode)
       char * path;
       int  mode;

   where:
       path      Address of a pathname
       mode      Protection mode of the new file

   returns: a channel number or -1
EXAMPLE:
   int x;
   if((x = creat("myfile", 0666)) == -1){
            perror("creat failed");
            exit(1);
   }
```

Basic IO (Cont'd)

```
Remove a file object from a directory
SYNOPSIS
       #include <unistd.h>
       int  unlink (path)
       char * path;

   where:
       path      Address of a pathname

   returns: 0 on success or -1
EXAMPLE:
   int x;
   if((x = unlink("myfile")) == -1){
            perror("unlink failed");
            exit(1);
   }
```

Basic IO (Cont'd)

```
Open or create and open a file
SYNOPSIS
     #include <fcntl.h>

     int  open (path, open_flag, protection_mode)
     char * path;
     int  open_flag;
     int  protection_mode;

   where:
     path             Address of a pathname
     open_flag        Open intent/open behavior flags
     protection_mode  rwx bits, if file is created
         mode defined constants:
         O_RDONLY read only
         O_WRONLY write only
         O_RDWR read and write

         O_NDELAY    no block on open
         O_NONBLOCK  no block on open
         O_APPEND    write at EOF
         O_CREAT     create if no file
         O_TRUNC     trash bytes in file
         O_EXCL      error if file exists
         O_SYNC      write-through buffer

   returns: a channel number or -1
```

Basic IO (Cont'd)

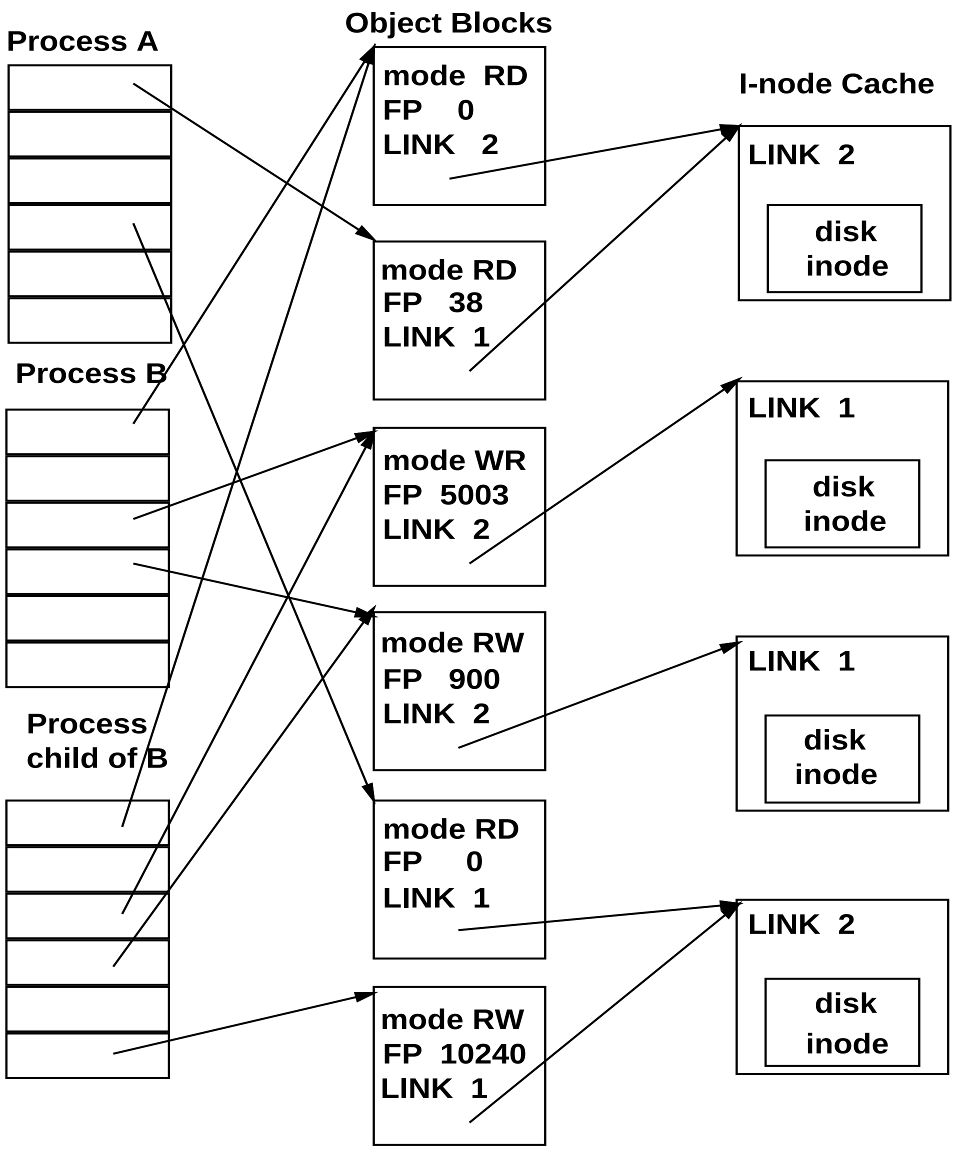
```
open()    (cont'd)

EXAMPLE:
   int fd;
   if((fd = open("myfile", O_RDWR|O_CREAT, 0666)) == -1){
            perror("open failed");
            exit(1);
   }

Close a file object
SYNOPSIS
       int  close (fildes)
       int  fildes;


   where:
       fildes    A valid, active file descriptor
   returns:  0 on success or -1
EXAMPLE:
   int x;
   if((x = close(fd)) == -1){
            perror("close failed");
            exit(1);
   }
```

Basic IO Kernel Connections



Basic IO (Cont'd)

```
Write bytes to a file object
Read bytes from a file object
SYNOPSIS
       #include <unistd.h>
       int      write (fildes, buffer, nbyte)
       int      fildes;
       char     buffer[];
       unsigned nbyte;

       int      read (fildes, buffer, nbyte)
       int      fildes;
       char     buffer[];
       unsigned nbyte;

   where:
       fildes    An active, valid file descriptor.
       buffer    User data buffer.
       nbyte     Size (bytes) of the read/write request.

   returns:  number of bytes read/written or -1
```

Basic IO (Cont'd)

```
read() , write()    (cont'd)

EXAMPLE:
   int fd, ret, numbytes=256;
   char *buf[256];
   if((fd = open("myfile", O_RDWR, 0)) == -1){
            perror("open failed");
            exit(1);
   }

   if((ret = write(fd, buf, numbytes)) == -1){
            perror("write failed");
            exit(1);
   }

   if((ret = read(fd, buf, numbytes)) == -1){
            perror("read failed");
            exit(1);
   }
```

Basic IO (Cont'd)

```
Seek a position in a file
SYNOPSIS
    #include <sys/file.h>
    #include <sys/types.h>
    #include <unistd.h>

    off_t  lseek (fildes, offset, whence)
    int    fildes;
    off_t  offset;
    int    whence;

   where:
    fildes  is the pointer to be changed: a valid, active file
            descriptor having a current position attribute.
    offset  is the new position of the file pointer.
    whence  is one of these three values specifying whether
            offset is an absolute or incremental address:
        SEEK_SET  Set fildes to offset bytes.
        SEEK_CUR  Set fildes to (fildes + offset) bytes.
        SEEK_END  Set fildes to (sizeof(fildes) + offset) bytes.

   returns:  file position or -1
EXAMPLE:
   int fd, ret;
   if((ret = lseek(fd, -45, SEEK_CUR)) == -1){
            perror("lseek failed");
            exit(1);
   }
   printf("file at %d byte location\n", ret);
```

Extended IO

```
Make a file, directory, FIFO or device
SYNOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>

       int  mknod (path, mode, device)
       char * path;
       int  mode;
       int  device;

   where:
       path      Address of a pathname
       mode      Access mode of the new file
       device    Device specifier

   returns:  0 on success or -1
EXAMPLE:
   int x;
   union{
     int  devcode;
     struct{
        short major,minor;
     }dev_parts;
   }dev;
   device.dev_parts.major = 3;
   device.dev_parts.minor = 1;
   if((x = mknod("/dev/mydev", S_IFCHRS|0600, dev.devcode)) == -1){
            perror("mknod type c failed");
            exit(1);
   }
```

Extended IO (Cont'd)

```
Make a hard link name
Make a symbolic link file object
SYNOPSIS
       #include <unistd.h>

       int  link (old_path, new_path)
       char * old_path;
       char * new_path;

       int  symlink (old_path, new_path)
       char * old_path;
       char * new_path;

   where:
       old_path  is a pathname to an existing file.
       new_path  is the additional pathname to be
                 assigned to the file.

   returns:  0 on success or -1
```

Extended IO (Cont'd)

`link() , symlink()` (cont'd)

```
EXAMPLE:
   int x;
   if((x = link("myfile", "sub1/myfiletoo")) == -1){
            perror("hard link failed");
            exit(1);
   }

   chdir("sub2");
   if((x = symlink("../myfile", "myfiletoo")) == -1){
            perror("symbolic link failed");
            exit(1);
   }

     % ls -li myfile
 74148 -rw-r--r--  2 bill  3961 May 12 1996 myfile

     % cd sub1; ls -l myfiletoo
 74148 -rw-r--r--  2 bill  3961 May 12 1996 myfiletoo

     % cd ../sub2; ls -l myfiletoo
 87655 lrwxrwxrwx  1 bill     8 Sep 14 20:48
                              myfiletoo -> ../myfile
```

Extended IO (Cont'd)

```
Change access mode to a file object
SYNOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>

       int  chmod (path, mode)
       char * path;
       int  mode;

       int  fchmod (fildes, mode)
       int  fildes;
       int  mode;

   where:
       path      Address of a pathname
       fildes    File descriptor
       mode      File's new mode

   returns:  0 on success or -1
```

Extended IO (Cont'd)

```
chmod() , fchmod()   (cont'd)

EXAMPLE:
   int x, fd;
   if((x = chmod("myfile", 0666)) == -1){
            perror("chmod failed");
            exit(1);
   }

   if((fd = open("myfile", O_RDONLY, 0)) == -1){
            perror("open failed");
            exit(1);
   }

   if((x = fchmod(fd, 0666)) == -1){
            perror("fchmod failed");
            exit(1);
   }
```

Extended IO (Cont'd)

```
Change ownership of a file object
Change ownership of a symlink
SYNOPSIS
       #include <unistd.h>

       int  chown (path, user, group)
       char * path;
       int  user;
       int  group;

       int  fchown (fildes, user, group)
       int  fildes;
       int  user;
       int  group;

       int  lchown (path, user, group)
       char * path;
       int  user;
       int  group;

   where:
       path   is the pathname of the file whose access is
              to be changed.
       user   is the new user id (st_uid) for the file. A
              value of -1 leaves the user id unchanged.
       group  is the new group id (st_gid) for the file.
              A value of -1 leaves the group id unchanged.
   returns:  0 on success or -1
EXAMPLE:
   int x, fd;
   if((x = chown("myfile", 215, 102)) == -1){
           perror("chmod failed");
           exit(1);
   }
```

Extended IO (Cont'd)

```
Check REAL access privilege to a file object
SYNOPSIS
       #include <sys/file.h>

       int  access (path, amode)
       char * path;
       int  amode;

   where:
       path      Address of a pathname naming a file
                 of type ordinary, directory, FIFO,
                 block special, character special,
                 or symbolic link.

       amode     Access mode bit pattern

   returns:  0 on success or -1
EXAMPLE:
   int x;
   if((x = access("myfile", W_OK|R_OK)) == -1){
            perror("no REAL access");
            exit(1);
   }
```

Extended IO (Cont'd)

```
Get file object status from i-node
SYNOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>

       int  stat (path, buffer_ptr)
       char * path;
       struct stat * buffer_ptr;

       int    fstat (fildes, buffer_ptr)
       int    fildes;
       struct stat * buffer_ptr;

   where:
       path           Address of a pathname
       fildes         A valid, active file descriptor
       buffer_ptr     Address of a stat buffer to fill

   returns:  0 on success or -1
```

Extended IO (Cont'd)

```
stat() , fstat()    (cont'd)

EXAMPLE:
       struct  stat
       {
            dev_t        st_dev;     /* device i-node on */
            ino_t        st_ino;     /* this i number    */
            mode_t       st_mode;    /* protection bits  */
            nlink_t      st_nlink;   /* hard link count  */
            uid_t        st_uid;     /* owner UID     */
            gid_t        st_gid;     /* owner GID     */
            dev_t        st_rdev;    /* device codes if device */
            off_t        st_size;    /* total bytes     */
            time_t       st_atime;   /* last file access */
            time_t       st_mtime;   /* last file modify */
            time_t       st_ctime;   /* last i-node modify */
            long         st_blksize; /*  element size   */
            long         st_blocks;  /* blocks allocated */
       };

   int x;
   struct stat buf;
   if((x = stat("myfile", &buf)) == -1){
            perror("stat failed");
            exit(1);
   }
   printf(" myfile is %d bytes long\n", buf.st_size);
```

Extended IO (Cont'd)

```
Change operations on an open file
SYNOPSIS
       #include <fcntl.h>

       int  fcntl (fildes, command, argument)
       int  fildes;
       int  command;
       int  argument;

   where:
       fildes     A valid, active file descriptor

       command    A file control command

       argument   An argument, either an integer
                  (when command is one of F_DUPFD,
                  F_GETFD, F_SETFD, or F_SETFL) or
                  a pointer to a struct flock (when
                  command is one of F_GETLK, F_SETLK,
                  F_SETLKW or F_FREESP)

   returns:  value, channel or -1
```

Extended IO (Cont'd)

```
fcntl()    (cont'd)

EXAMPLE:
   defined cmd values:
    F_DUPFD    dup channel on lowest channel
               greater than or equal to arg
    F_GETFD    get channel close-on-exec flag
               (LSB 0 = keep-open-across-exec)
    F_SETFD    set close-on-exec flag as above
               using arg (0 open, 1 close)
    F_GETFL    get channel status flag
    F_SETFL    set channel status flag with arg
    F_FREESP   free storage space on a section
               of the file using flock structure
    F_SETLK    set or clear a lock using flock
    F_SETLKW   synchronous version of F_SETLK
    F_GETLK    check for lock status using flock

   struct flock
   {
    short   l_type;   /* F_RDLCK, F_UNLCK, F_WRLCK */
    short   l_whence; /* SEEK_SET, SEEK_CUR, SEEK_END */
    off_t   l_start;  /* relative offset to start */
    off_t   l_len;    /* length to lock (0 to EOF) */
    pid_t   l_pid;    /* pid of process with lock */
    short   l_sysid;  /* system_id of process with lock */
   };
```

Extended IO (Cont'd)

`fcntl()    (cont'd)`

```
   defined arg values for F_GETFL and F_SETFL:
    O_APPEND    change to append mode
    O_NONBLOCK  non_blocking mode for terminals and
                pipes
    O_SYNC      synchronous writes

   int x, fd, flag;
   struct flock mylock;
   if((flag = fcntl(0, F_GETFL, 0)) == -1){
            perror("fcntl F_GETFL failed");
            exit(1);
   }

   if((x = (fcntl(0, F_SETFL, flag|O_NONBLOCK)) == -1){
            perror("fcntl F_SETFL failed");
            exit(1);
   }

   mylock.l_type = F_WRLCK;
   mylock.l_whence = SEEK_SET;
   mylock.l_start = 300;
   mylock.l_len = 0;
   if((x = (fcntl(fd, F_SETLKW, &mylock) == -1){
            perror("fcntl F_SETLKW failed");
            exit(1);
   }
```