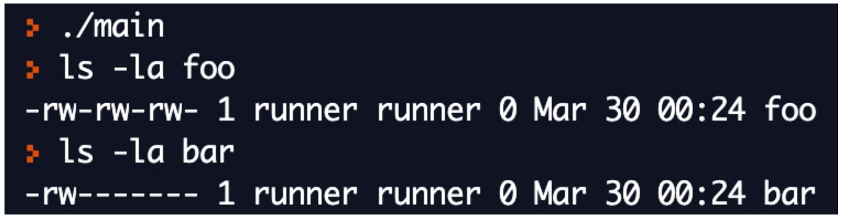
File System Calls – Part 2

umask

- #include <sys/types.h>
  #include <sys/stat.h>
  mode_t umask(mode_t *mask*);
  - Set file mode creation *mask* and return the old value.
  - When creating a file, permissions are turned off if the corresponding bits in *mask* are set.
  - Return value
    - This system call always succeeds and the previous value of the mask is returned.
  - cf. "umask" shell command

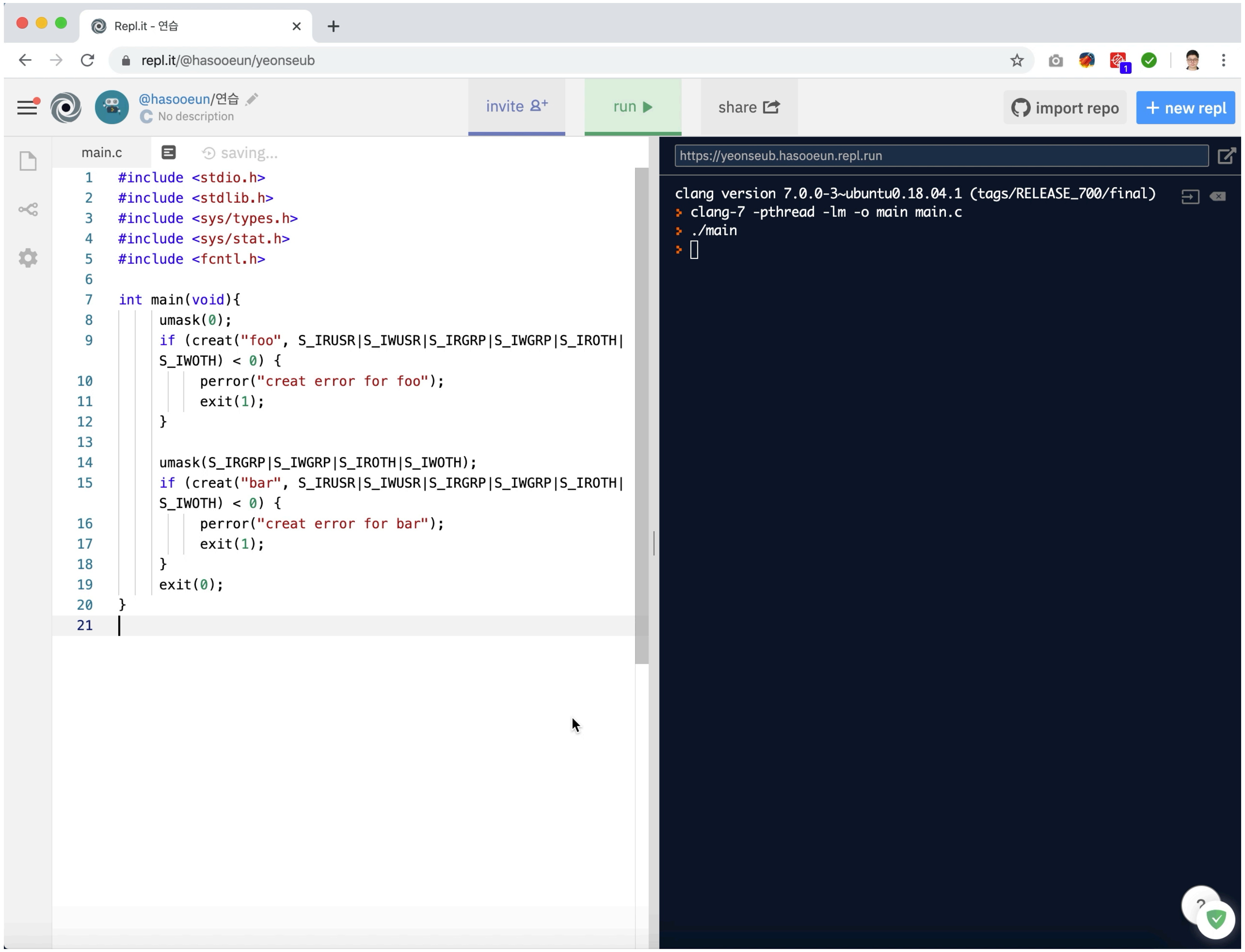
Example #5: umask

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(void) {
  umask(0);
  if (creat("foo", S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH) < 0) {
    perror("creat error for foo");
    exit(1);
  }

  umask(S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH);
  if (creat("bar", S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH) < 0) {
    perror("creat error for bar");
    exit(1);
  }
  exit(0);
}
```



```
> ./main
> ls -la foo
-rw-rw-rw- 1 runner runner 0 Mar 30 00:24 foo
> ls -la bar
-rw------- 1 runner runner 0 Mar 30 00:24 bar
```

Example #5: umask



```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(void){
    umask(0);
    if (creat("foo", S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|
    S_IWOTH) < 0) {
        perror("creat error for foo");
        exit(1);
    }

    umask(S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH);
    if (creat("bar", S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|
    S_IWOTH) < 0) {
        perror("creat error for bar");
        exit(1);
    }
    exit(0);
}
```

```
clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
> clang-7 -pthread -lm -o main main.c
> ./main
> 
```

access (1)

- #include <unistd.h>

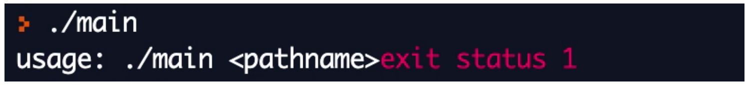
  int access(const char **pathname*, int *mode*);
  - Checks whether the process would be allowed to read, write or test for existence of the file whose name is *pathname*.
  - If pathname is a symbolic link, permissions of the file referred to by this symbolic link are tested.
  - The check is done with the process's real uid and gid. (neither effective uid nor effective gid)

access (2)

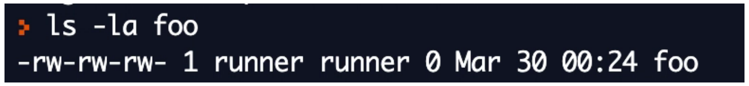
- *mode*
  - R_OK
    - file exist, read permission
  - W_OK
    - file exist, write permission
  - X_OK
    - file exist, execution permission
  - F_OK
    - file exist
- Return value
  - On success zero is returned. On error -1 is returned.
  - Returns an error if any of the access types in the requested call fails, even if other types might be successful.
  - Ex: access(“bit”, R_OK|W_OK) checks read and write permission.

Example #6: access

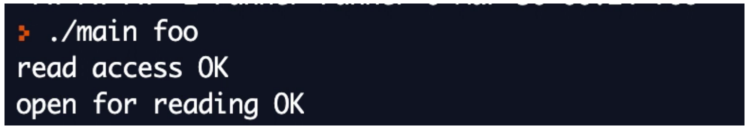
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
        if (argc != 2) {
                fprintf(stderr, "usage: a. out <pathname>");
                exit(1);
        }
        if (access(argv[1], R_OK) < 0)
                perror("access error");
        else
                printf("read access OK\n");
        if (open(argv[1], O_RDONLY) < 0)
                perror("open error");
        else
                printf("open for reading OK\n");
        exit(0);
}
```



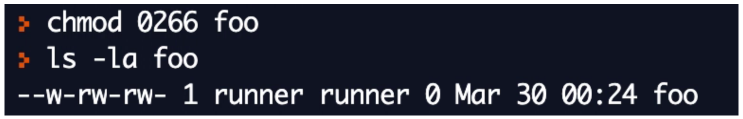
```
❯ ./main
usage: ./main <pathname>exit status 1
```



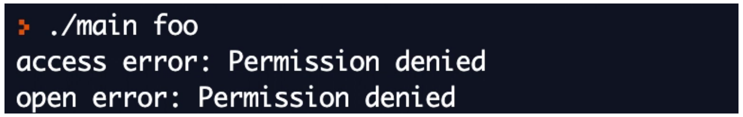
```
❯ ls -la foo
-rw-rw-rw- 1 runner runner 0 Mar 30 00:24 foo
```



```
❯ ./main foo
read access OK
open for reading OK
```

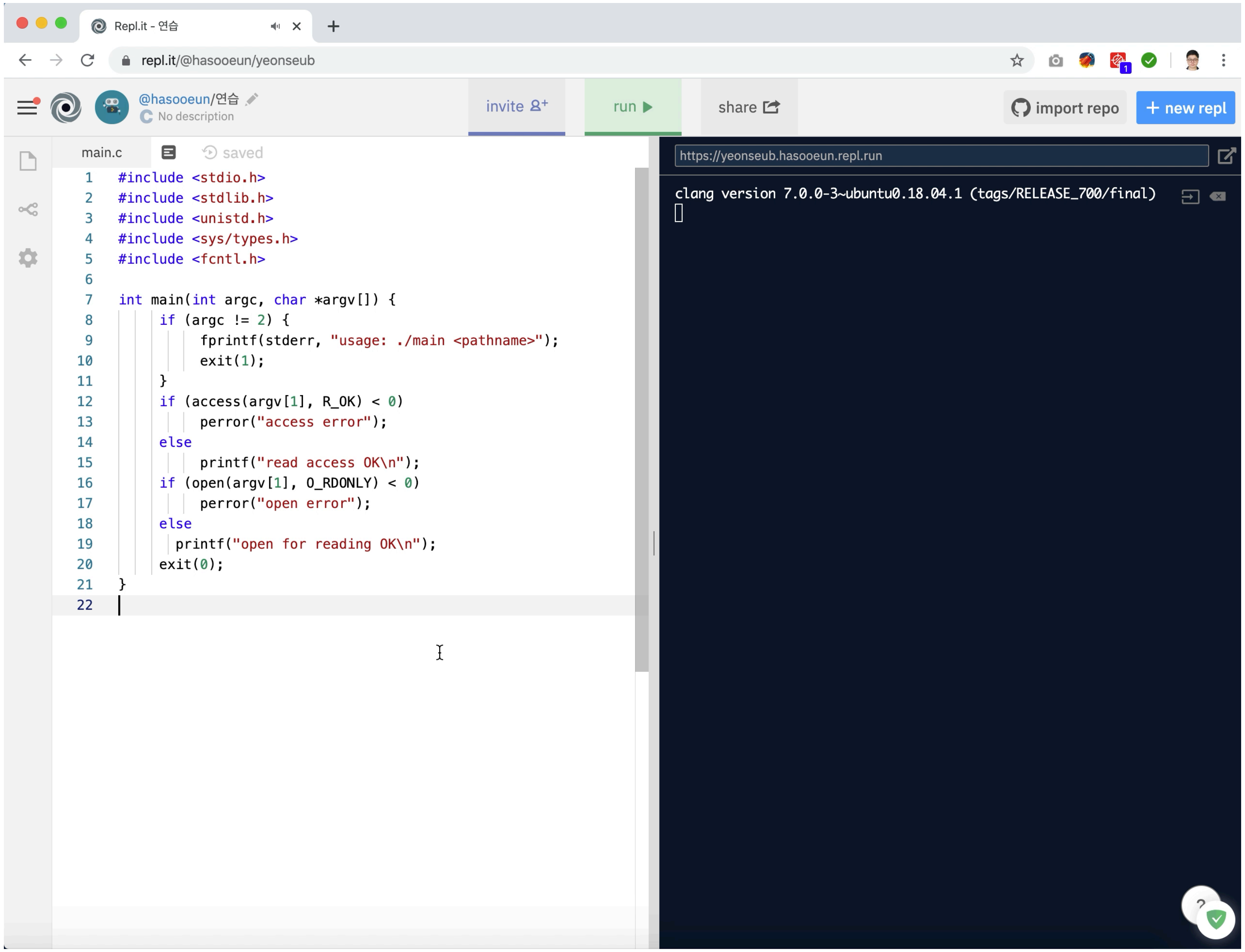


```
❯ chmod 0266 foo
❯ ls -la foo
--w-rw-rw- 1 runner runner 0 Mar 30 00:24 foo
```



```
❯ ./main foo
access error: Permission denied
open error: Permission denied
```

Example #6: access



```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: ./main <pathname>");
        exit(1);
    }
    if (access(argv[1], R_OK) < 0)
        perror("access error");
    else
        printf("read access OK\n");
    if (open(argv[1], O_RDONLY) < 0)
        perror("open error");
    else
      printf("open for reading OK\n");
    exit(0);
}
```

```
clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
```

stat, fstat, lstat

- #include <sys/stat.h>
  #include <unistd.h>
  int stat(const char **file_name*, struct stat **buf*);
  int fstat(int *fd*, struct stat **buf*);
  int lstat(const char **file_name*, struct stat **buf*);
  - Return information about the specified file.
  - Do not need any access rights to the file. But need search rights to all directories named in the path leading to the file.
  - stat( ), fstat( )
    - Stats the file pointed to by *file_name* or by *fd* and fills in *buf.*
  - lstat( )
    - Same as stat( ) except that the symbolic link is stated itself (i.e. do not follow the link).

Structure stat

```
struct stat {
    dev_t st_dev;                   /* device number */
    ino_t st_ino;                   /* inode number */
    mode_t st_mode;                 /* file type, mode (permissions) */
    nlink_t st_nlink;               /* number of hard links */
    uid_t st_uid;                   /* user ID of owner */
    gid_t st_gid;                   /* group ID of owner */
    dev_t st_rdev;                  /* device type for special files (if inode device) */
    off_t st_size;                  /* total size, in bytes */
    unsigned long st_blksize;       /* blocksize for I/O */
    unsigned long st_blocks;        /* # of blocks allocated */
    time_t st_atime;                /* time of last access */
    time_t st_mtime;                /* time of last modification */
    time_t st_ctime;                /* time of last status change */
};
```

stat Fields

- st_blocks
  - The number of blocks allocated to this file.
- st_blksize
  - The “preferred” block size for efficient file system I/O.
  - Writing to a file in smaller chunks may cause an inefficient *read-modify-rewrite*.
- st_atime
  - Changed by mknod( ), utime( ), read( ), write( ), and truncate( ).
- st_mtime
  - Changed by mknod( ), utime( ), and write( ).
- st_ctime
  - Changed by writing or by setting inode information.
  - owner, group, link count, mode, etc.

POSIX Macro for File Type

- S_ISLNK(*st_mode*): symbolic link
- S_ISREG(*st_mode*): regular file
- S_ISDIR(*st_mode*): directory
- S_ISCHR(*st_mode*): character device
- S_ISBLK(*st_mode*): block device
- S_ISFIFO(*st_mode*): fifo (named pipe)
- S_ISSOCK(*st_mode*): socket (unix domain socket)

POSIX Flags for st_mode Field

S_IFMT: 0170000	bit mask for the file type bit fields.
S_IFSOCK:0140000	socket.
S_IFLNK: 0120000	symbolic link.
S_IFREG: 0100000	regular file
S_IFBLK: 0060000	block device
S_IFDIR: 0040000	directory
S_IFCHR: 0020000	character device
S_IFIFO: 0010000	fifo
S_ISUID: 0004000	set-user-id bit (POSIX)
S_ISGID: 0002000	set-group-id bit (POSIX)
S_ISVTX: 0001000	sticky bit

file type (4bit)	special(3bit)	permission(9bit)

Usage: stat, fstat, lstat

- Return value
  - On success, zero is returned.
  - On error, -1 is returned.

- Example

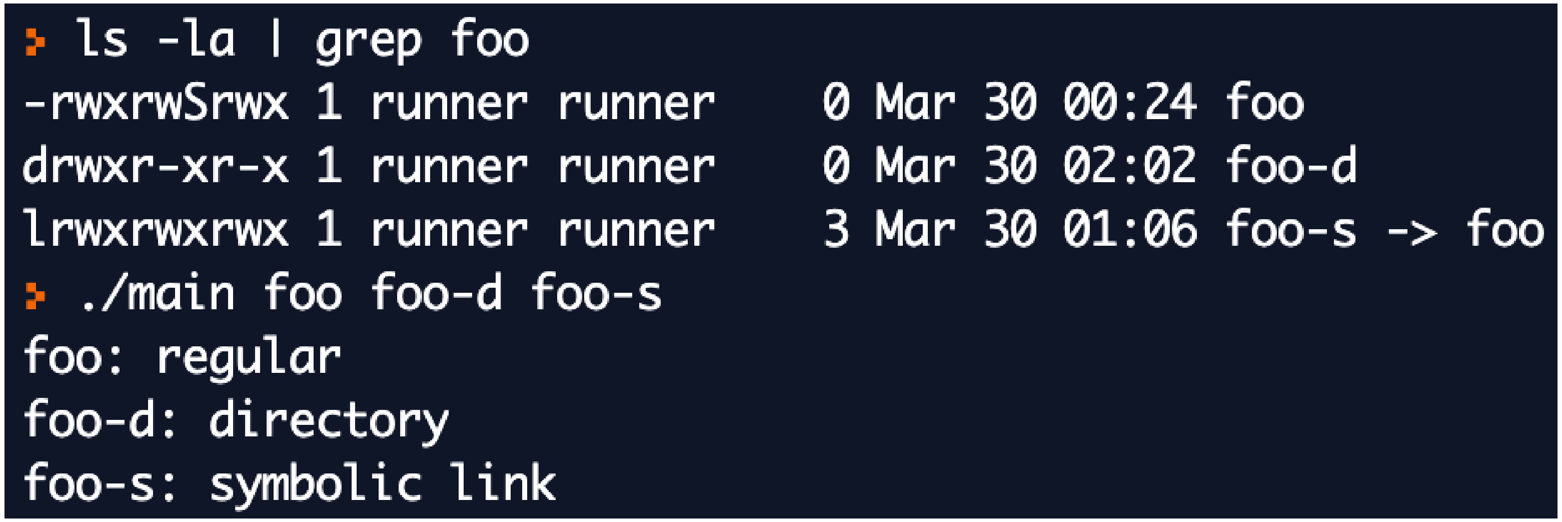
```
...
struct stat buf;

stat("/etc/passwd", &buf);
if (buf.st_mode & S_IRUSR) {
...
}
```

Example #7: lstat (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>

int main(int argc, char *argv[]) {
  int i;
  struct stat buf;
  char *ptr;
  for (i = 1; i < argc; i++) {
    printf("%s: ", argv[i]);
    if (lstat(argv[i], &buf) < 0) {
      perror("lstat error");
      continue;
    }
    if (S_ISREG(buf.st_mode)) ptr = "regular";
    else if (S_ISDIR(buf.st_mode)) ptr = "directory";
    else if (S_ISCHR(buf.st_mode)) ptr = "charactor special";
    else if (S_ISBLK(buf.st_mode)) ptr = "block special";
    else if (S_ISFIFO(buf.st_mode)) ptr = "fifo";

```

Example #7: lstat (2)

```c
#ifdef S_ISLNK
    else if (S_ISLNK(buf.st_mode)) ptr = "symbolic link";
#endif
#ifdef S_ISSOCK
    else if (S_ISSOCK(buf.st_mode)) ptr = "socket";
#endif
    else ptr = "** unkown mode **";
    printf("%s\n", ptr);
  }
  exit(0);
}
```



```
> ls -la | grep foo
-rwxrwSrwx 1 runner runner    0 Mar 30 00:24 foo
drwxr-xr-x 1 runner runner    0 Mar 30 02:02 foo-d
lrwxrwxrwx 1 runner runner    3 Mar 30 01:06 foo-s -> foo
> ./main foo foo-d foo-s
foo: regular
foo-d: directory
foo-s: symbolic link
```

chmod and fchmod

- #include <sys/types.h>

  #include <sys/stat.h>

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

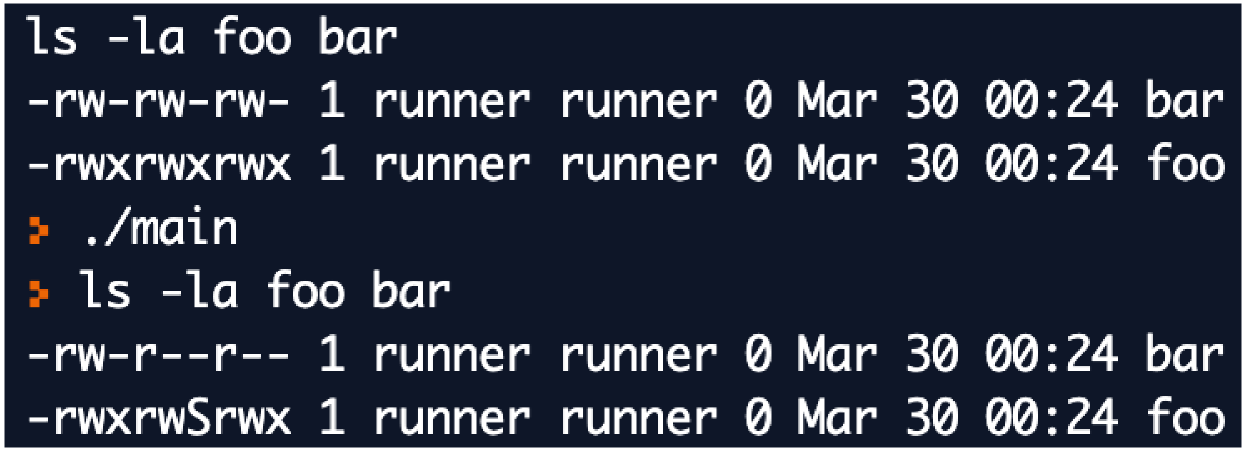
  int fchmod(int *fd*, mode_t *mode*);
  - Change permissions of a file.
  - The mode of the file given by *path* or referenced by *fd* is changed.
  - *mode* is specified by OR'ing the following.
    - S_ISUID, S_ISGID, S_ISVTX, S_I{R,W,X}{USR,GRP,OTH}
  - On success, zero is returned. On error, -1 is returned.

Example #8: chmod (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>

int main(void) {
  struct stat statbuf;

  if (stat("foo", &statbuf) < 0) {
    perror("stat error for foo");
    exit(1);
  }

  /* turn on set-group-ID and turn off group-execute */
  if (chmod("foo", (statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0) {
    perror("chmod error for foo");
    exit(1);
  }


```

Example #8: chmod (2)

```
  /* set absolute mode to "rw-r--r--" */
  if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0) {
    perror("chmod error for bar");
    exit(1);
  }
  exit(0);
}
```



```
ls -la foo bar
-rw-rw-rw- 1 runner runner 0 Mar 30 00:24 bar
-rwxrwxrwx 1 runner runner 0 Mar 30 00:24 foo
› ./main
› ls -la foo bar
-rw-r--r-- 1 runner runner 0 Mar 30 00:24 bar
-rwxrwSrwx 1 runner runner 0 Mar 30 00:24 foo
```

chown, fchown, lchown

- #include <sys/types.h>
  #include <unistd.h>
  int chown(const char **path*, uid_t *owner*, gid_t *group*);
  int fchown(int *fd*, uid_t *owner*, gid_t *group*);
  int lchown(const char **path*, uid_t *owner*, gid_t *group*);
  - The owner of the file specified by *path* or by *fd* is changed.
  - Only the superuser and the current owner may change the owner of a file.
  - The owner of a file may change the group of the file to any group of which that owner is a member.
  - When the owner or group of an executable file are changed by a non-superuser, the S_ISUID and S_ISGID mode bits are *cleared*.
  - On success, zero is returned. On error, -1 is returned.
  - *lchown()* is same as *chown( )* except that the symbolic link is stated itself (i.e. do not follow the link).

truncate and ftruncate

- #include <unistd.h>

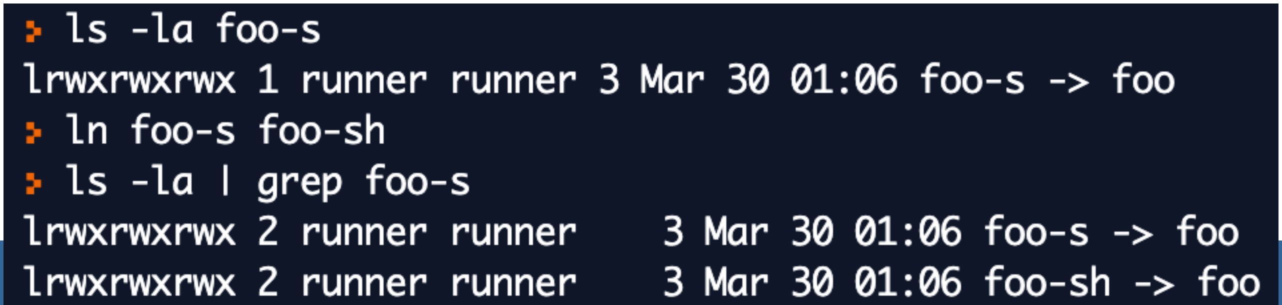
  int truncate(const char **path*, off_t *length*);

  int ftruncate(int *fd*, off_t *length*);
  - Causes the file referenced by a path or a file descriptor to have a size of *length* bytes.
  - If the file previously was larger than *length*, the extra data is lost.
  - If the file size was less than *length*, the effect is system dependent.
  - Return value
    - On success, zero is returned.
    - On error, -1 is returned.

link

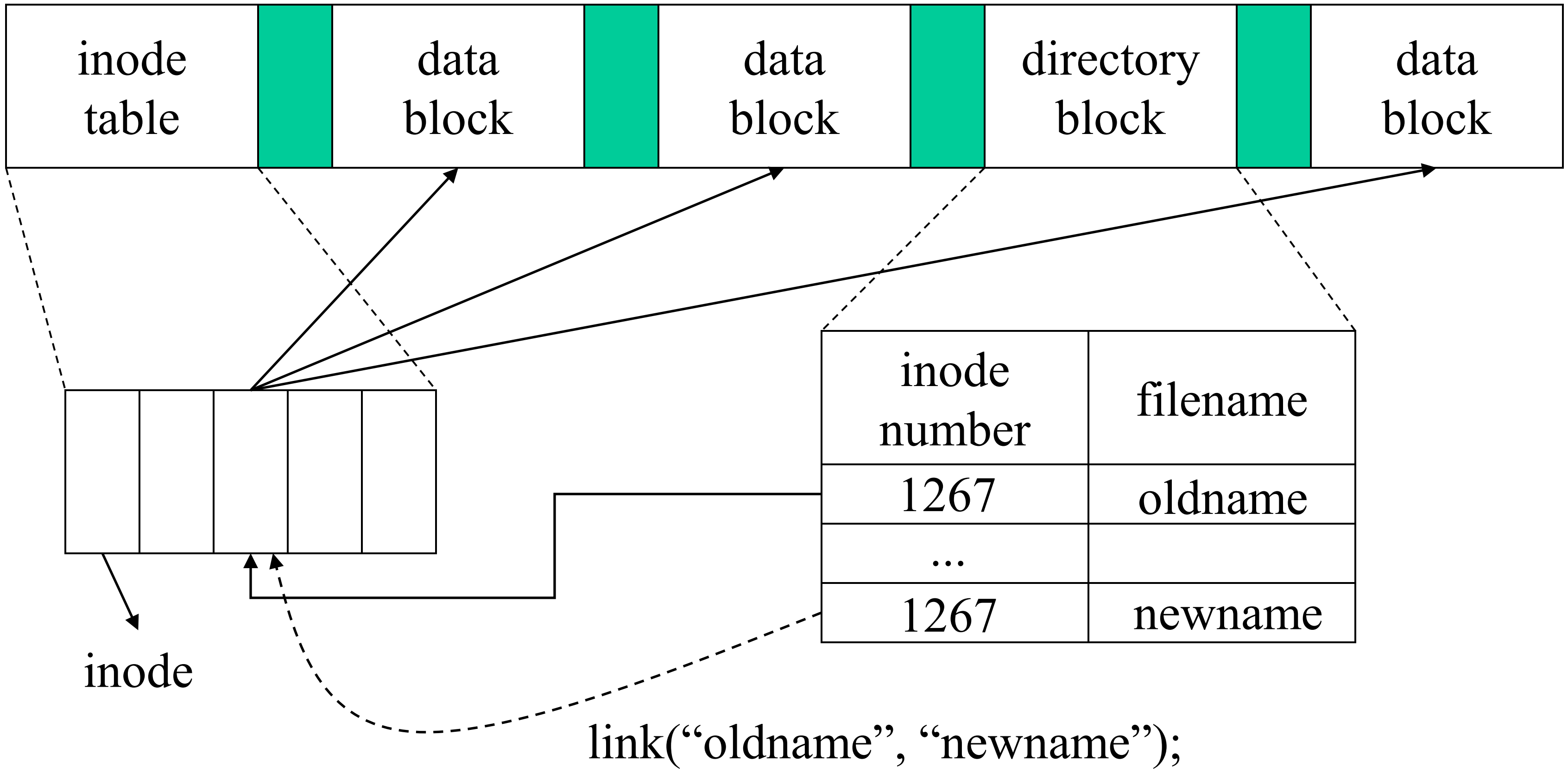
- #include <unistd.h>

  int link(const char **oldpath*, const char **newpath*);
  - Make a new name for a file
    - Creates a new link (also known as a *hard link*) to an existing file.
  - If *newpath* exists it will not be overwritten.
  - This new name may be used exactly as the old one for any operation.
    - Both names refer to the same file.
    - Have the same permissions and ownership.
    - It is impossible to tell which name was the original.
  - Hard links, as created by link, cannot span file systems.



```
› ls -la foo-s
lrwxrwxrwx 1 runner runner 3 Mar 30 01:06 foo-s -> foo
› ln foo-s foo-sh
› ls -la | grep foo-s
lrwxrwxrwx 2 runner runner    3 Mar 30 01:06 foo-s -> foo
lrwxrwxrwx 2 runner runner    3 Mar 30 01:06 foo-sh -> foo
```

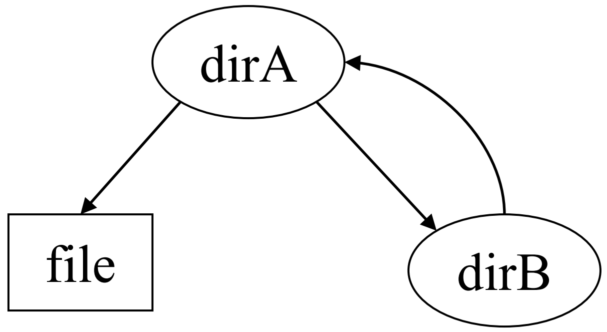
View of link



inode table
data block
data block
directory block
data block
inode number
filename
1267
oldname
...
1267
newname
inode
link("oldname", "newname");

Directory link

- Only the superuser can create a hard link to a directory.



- Return value
  - On success, zero is returned.
  - On error, -1 is returned.

unlink

- #include <unistd.h>

  int unlink(const char **pathname*);
  - Delete a name and possibly the file it refers to
    - Deletes a name from the file system.
    - If that name was the last link to a file and no processes have the file open, the file is deleted.
    - If the name was the last link to a file but any processes still have the file open, the file will remain in existence until the last file descriptor referring to it is closed.
    - If the name referred to a symbolic link, the link is removed.
  - Return value
    - On success, zero is returned. On error, -1 is returned.

rename

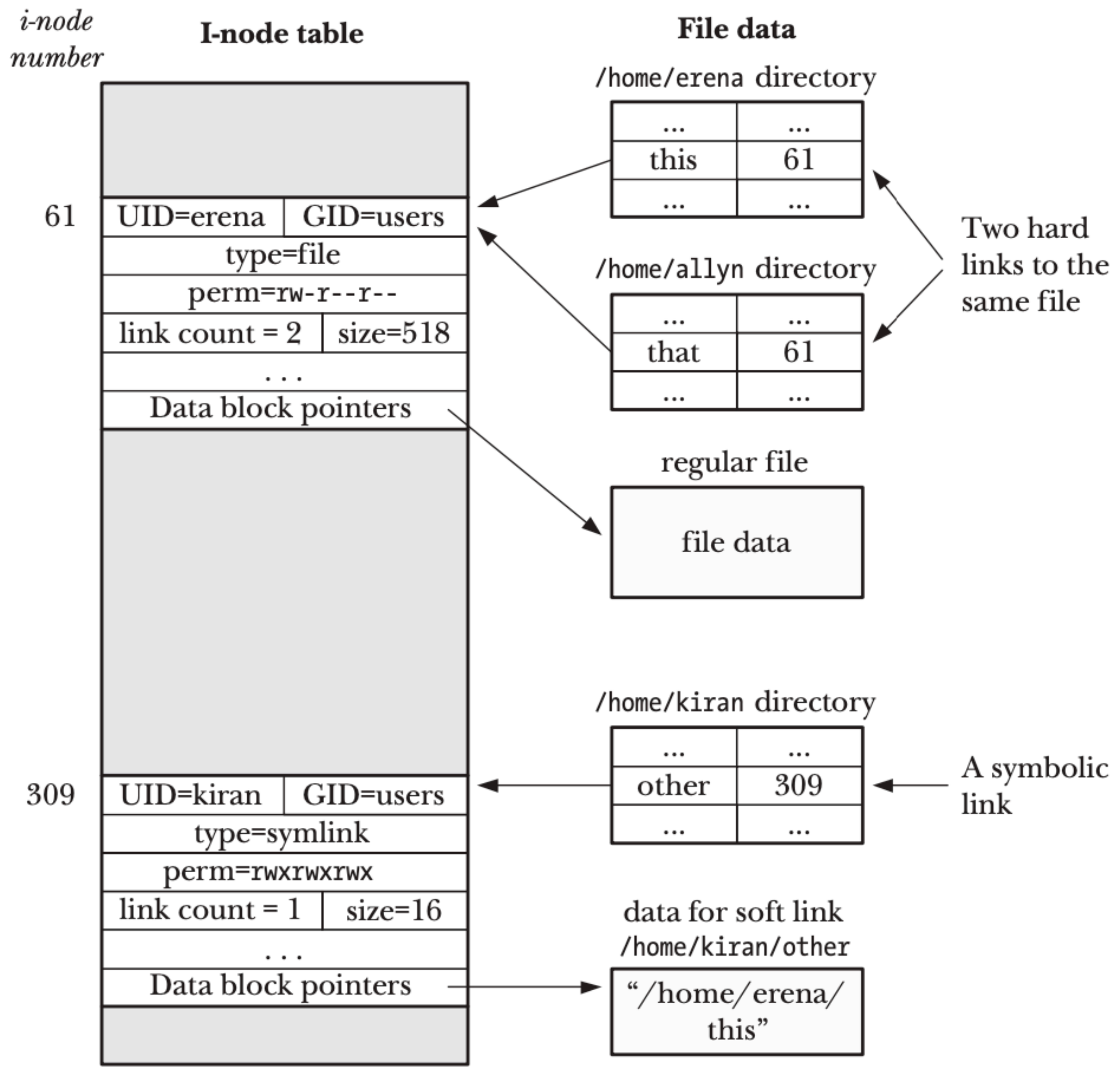
- #include <stdio.h>

  int rename(const char **oldpath*, const char **newpath*);
  - Changes the name of *oldpath* to *newpath*.
  - If *oldpath* names open nonexistent file, or if *newpath* names a file that already exists, then the action of rename( ) is *implementation-dependent*.
  - Return value
    - On success, zero is returned.
    - On error, -1 is returned.

Symbolic Link

- Symbolic link is indirect pointer to a file.
  - Hard link points directly to the inode of the file.
  - Limitation of hard links
    - Require that the link and the file reside in the same file system.
    - Only the superuser can create a hard link to a directory.
  - There are no file system limitations on a symbolic link.
  - Anyone can create a symbolic link to a directory.

Symbolic Link



symlink

- #include <unistd.h>
  int symlink(const char **oldpath*, const char **newpath*);
  - Creates a symbolic link named *newpath* which contains the string *oldpath*.
  - Symbolic links are interpreted at run-time.
  - Dangling link
    - May point to an non-existing file.
  - The permissions of a symbolic link are irrelevant.
    - The ownership is ignored when following the link.
    - Permission is checked when removal or renaming of the link is requested and the link is in a directory with the sticky bit set.
  - If *newpath* exists it will not be overwritten.
  - Return value
    - On success, zero is returned. On error, -1 is returned.

readlink

- #include <unistd.h>

  int readlink(const char **path*, char **buf*, size_t *bufsiz*);
  - Read value of a symbolic link (does *not* follow the link).
    - Places the contents of the symbolic link *path* in the buffer *buf*, which has size *bufsiz.*
  - Return value
    - The count of characters placed in the buffer if it succeeds.
    - -1 if an error occurs.

Thank you