Review

- Command Line Argument
- sat, fsat, lsat system Call
- ID's for a process
- File Access permission
- access System Call

Preview

- umask() system call
- chmod(), fchmod() system call
- File truncation with truncate()
- File system in Linux
- link(), unlink() system calls
- remove() and rename() system calls

A System Call umask

- Nine permission bits are associated with a file.
- A umask system call set the file mode creation mask for the process and return the previous value
- Prototype

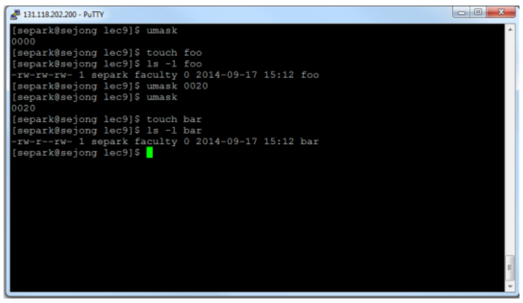
```
#include <sys/types.h>
#include <sys/stat.h>
mode_t umask(mode_t cmask);
                        Returns: previous file mode creation
```

- Most user does not deal with umask value.
- When writing a programs that create new files, if we want to assure that specific access permission bits are not enabled, we must modify the umask value while the process is running.

A System Call umask()

- We can check current mask value with shell command `umask`. It shows current mask value for file creation.
- If the mask is cleared (0000) then we can create a file with any mode.
- But if mask is (0020), write protected for group. A file will be created without group write permission.
- `touch` shell command create a file with `rw-rw-rw` with cleared mask. But if mask is 0022, a file will be created with `rw-r—r--`.
- When writing a programs that create new files, if we want to assure that specific access permission, you must clear the file mode creation mask by umask() system call before creating a new file.

A System Call umask()



```
[separk@sejong lec9]$ umask
0000
[separk@sejong lec9]$ touch foo
[separk@sejong lec9]$ ls -l foo
-rw-rw-rw- 1 separk faculty 0 2014-09-17 15:12 foo
[separk@sejong lec9]$ umask 0020
[separk@sejong lec9]$ umask
0020
[separk@sejong lec9]$ touch bar
[separk@sejong lec9]$ ls -l bar
-rw-r--rw- 1 separk faculty 0 2014-09-17 15:12 bar
[separk@sejong lec9]$
```

```c
/********************************************************
  umask.c : change a file permission inside a process
********************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

void err_sys(char *str)
{
   printf ("%s\n",str);
   exit (1);
}
int main ()
{
   umask(0);
   if (creat("foo",S_IRUSR |S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH) <0)
     err_sys ("creat Error for foo ");
   umask(S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH); */ same as 0033 */
   if (creat ("bar",S_IRUSR |S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH) <0)
     err_sys ("creat Error for bar ");
   exit(0);
}
```

chmod, fchmod System Call

- These two function allow us to change the file access permissions for <u>an existing file</u>.
- `fchmod()` operate on a file that has already opened.

Prototype:

```
#include<sys/types.h>
#include<sys/stat.h>
int chmod (const char *pathname, mode_t mode)
int fchmod (int filedes, mode_t mode)

                         Return 0 if OK, - 1 on a error
```

chmod, fchmod System Call

- <u>To change the permission bit of a file</u>, <u>the effective user ID of the process must equal the owner of the file</u> or <u>process must have superuser permission</u>.
- Since chmod() <u>only update only the time that the i-node was last changed</u>, the time a file was modified will not change.

chmod, fchmod System Call

- The mode constants for chmod from <sys/sat.h>

mode	Description
S_ISUID	set-user-ID on execution
S_ISGID	set-group-ID on execution
S_ISVTX	saved-text (sticky bit)-used for using swap area
S_IRWXU	read, write, execute by user
S_IRUSR	read by user
S_IWUSR	write by user
S_IXUSR	execute by user
S_IRWXG	read, write, execute by group
S_IRGRP	read by group
S_IWGRP	write by group
S_IXGRP	execute by group
S_IRWXO	read, write, execute by other
S_IROTH	read by other
S_IWOTH	write by other
S_IXOTH	execute by other

```
//chmode.c demonstrate how to use fchmod() system call
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
void err_sys(char *str)
{
   printf ("%s",str);
   exit (1);
}
int main()
{
  int fd;
  struct stat info;
  umask(0);
  if ((fd = creat("test.file", 0660)) <0) //rw-rw----
      err_sys("Creat File Open Error");

  fstat(fd, &info);
  printf("original permissions were: %o\n", info.st_mode);
  if (fchmod(fd, 0770) != 0) //rwxrwx---
      err_sys("chmod() error");
  fstat(fd, &info);
  printf("after chmod(), permissions are: %o\n", info.st_mode);
  close(fd);
}
```

```
/*chmodex.c  demonstrate how to use chmod system call*/

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

void err_sys(char *str)
{
   printf ("%s\n",str);
   exit (1);
}
int main()
{
   struct stat buff;
   /*turn on set-group-ID and turn off group execute */
   if (stat ("foo", &buff) < 0)
      err_sys("stat error for foo");
   if (chmod ("foo", (buff.st_mode & -S_IXGRP) |S_ISGID) <0)
      err_sys("chomd error for foo");

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

chmod, fchmod System Call

Sticky bit : S_ISVTX

- If a sticky bit for an executable file is set, then the first time the program was executed, a copy of the executable file was saved into swap area when the process terminate.
- If this process become active again, it will be <u>loaded from the swap area</u> in the memory.
- Text editor, C compiler,…

chmod, fchmod System Call

- If we try to set the sticky bit of a regular file, the sticky bit in the mode is automatically turned off.
- Only super user can turn on sticky bit to prevent malicious users from setting the sticky bit and trying to fill up the swap area..

chown, fchown, lchown System Call

- The chown functions allows us to change the user ID and group ID of a file.
- Unix only allows super user to change the ownership of a file.
- Prototype:

```
#include <sys/types.h>
#include <unistd.h>
int chown (const char *pathname, uid_t owner, gid_t group);
int fchown (int filedes, uid_t owner, gid_t group);
/*change ownership of symbolic link) */
int lchown (const char *pathname, uid_t owner, gid_t group);

                                    Return 0 if OK, else return -1
```

File truncation with truncate()

- We can chopping off data at the end of file by using truncate() system call.

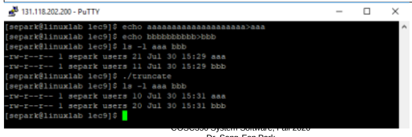
```
#include <sys/types.h>
#include <unistd.h>
int truncate (const char *pathname, off_t length);
int ftruncate (int fd, off_t length);
                            Return 0 if OK, else return -1
```

- If the previous size of the file was greater than length, the data beyond length is no longer accessible.
- If the previous size was less than length, the file size will increase and the data between the old end of file and the new end of file will read as 0

File truncation with truncate()

- Test with following program with two text file.
  - Text file "aaa" contains 20 character
  - Text file "bbb" contains 10 character

```
#include <unistd.h>
#include<sys/types.h>
int main()
{
          truncate ("aaa", 10);
          truncate ("bbb", 20);
          return 0;
}
```

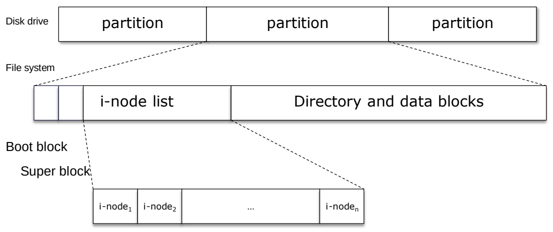


```
[separk@linuxlab lec9]$ echo aaaaaaaaaaaaaaaaaaaa>aaa
[separk@linuxlab lec9]$ echo bbbbbbbbbb>bbb
[separk@linuxlab lec9]$ ls -l aaa bbb
-rw-r--r-- 1 separk users 21 Jul 30 15:29 aaa
-rw-r--r-- 1 separk users 11 Jul 30 15:29 bbb
[separk@linuxlab lec9]$ ./truncate
[separk@linuxlab lec9]$ ls -l aaa bbb
-rw-r--r-- 1 separk users 10 Jul 30 15:31 aaa
-rw-r--r-- 1 separk users 20 Jul 30 15:31 bbb
[separk@linuxlab lec9]$
```

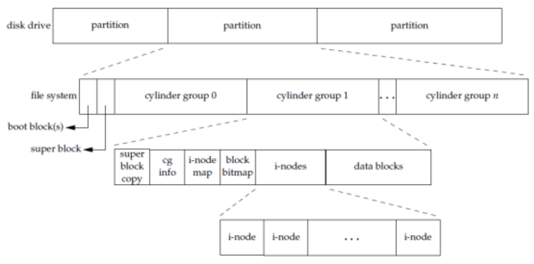
File System

- Various implementations of the UNIX file system are in use today.
- We can think of a disk drive being divided into one or more partitions. Each partition can contain a file system
- The i-nodes are fixed-length entries that contain most of the information about a file.
  - Attributes
  - Block addresses used to save a file.

File System



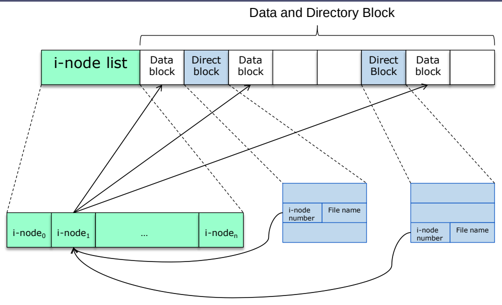
File System



File System

- Following figure shows the i-node and data block portion of partition in more detail.
- Two directory entries point to the same i-node entry.
- Every i-node has a link count that contains the number of directory entries that point to it. The link count is contained in the **st_nlink** in the **stat** data structure
- A file be deleted when the link count is 0.
- **Unlinking** means delete a file entry from a directory.
- If link count is > 1, unlinking file does not delete the file.

File System



File System

```
struct stat {
        mode_t st_mode;   /*file type & mode (permissions) */
        ino_t st_ino;     /* i-node number */
        dev_t st_dev;     /* device number (file system) */
        dev_t st_rdev;    /* device number for special files */
        nlink_t st_nlink; /* number of links */
        uid_t st_uid;     /* user ID of owner */
        gid_t st_gid;     /* group ID of owner */
        off_t st_size;    /* size in bytes, for regular files */
        time_t st_atime;  /* time of last access */
        time_t st_mtime;  /* time of last modification */
        time_t st_ctime;  /* time of last file status change */
        blksize_t st_blksize; /* best I/O block size */
        blkcnt_t st_blocks; /*number of 512 byte blocks allocated */
        mode_t st_attr;   /* The DOS-style attributes for this file */
};
```

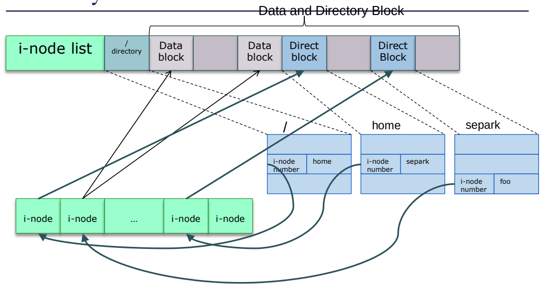
File System

- The i-node contains most of information (attributes) about the file: the file type, the file's access permission bit,…and so on.
- But only two information are stored in the directory entry: the filename and the i-node number.
- When move a file from one directory to another directory need only change directory entry point.

File System

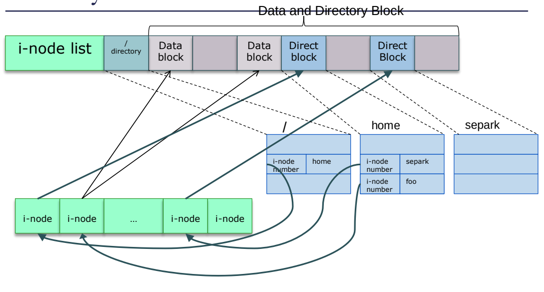
Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File System



Move /home/separk/foo to /home/foo

File System



Move /home/separk/foo to /home/foo

link, unlink, remove and rename

- Any file can have multiple directory entries pointing to its i-node.
- The way we can create a link to an existing file is with the link system call.
- Prototype:

```
#include <unistd.h>
int link (const char *existingpath, const char *newpath)

                  Return 0 if there is no error, else retun -1
```

link, unlink, remove and rename

```
// creatlink.c two path to create a new pass from different location
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
void err_sys(char *str)
{
   printf ("%s\n",str);
   exit (1);
}
//
int main (int argc, char *argv[])
{
   struct stat buff;

    if (argc < 2)
      err_sys ("Less than two argument Error");

     /* increase the number of link  by one */
     if (stat (argv[1], &buff) < 0)
                 err_sys("stat error for foo");
      printf ("link count for a file %s was %d \n",argv[1], buff.st_nlink);

      if (link (argv[1], argv[2]) <0)
                 err_sys("Link Error");

      if (stat (argv[1], &buff) < 0)
                 err_sys("stat error for foo");
      printf ("link count for a file %s is now %d \n",argv[1], buff.st_nlink);

   return 0;
}
```

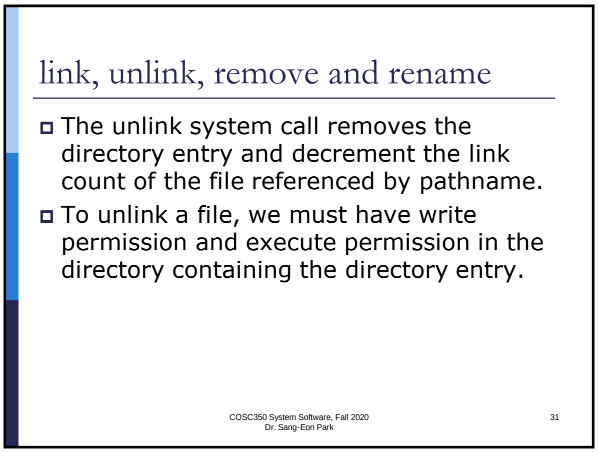
link, unlink, remove and rename

- The link() system call create a new directory entry *newpath* that references the existing file *existingpath*.
- The creation and increment of the link count be done automatically by kernel.
- Only a <u>superuser process can create a new link that points to a directory</u>. Because link system calls <u>can cause loops in the filesystem</u>.

link, unlink, remove and rename

- By using the unlink system call, <u>we can remove an existing directory entry</u>.
- Prototype:

```
#include <unistd.h>
int unlink (const char *pathname)
          return 0 if no error, return -1 if error
```



link, unlink, remove and rename

- The unlink system call removes the directory entry and decrement the link count of the file referenced by pathname.
- To unlink a file, we must have write permission and execute permission in the directory containing the directory entry.