14 Lecture: Filesystem wrapup for reals

Outline:

Announcements
Review: Unbuffered IO
Onwards: lseek(2)
  Structure of the filesystem
  From last time: Links, symbolic and otherwise
Manipulating the filesystem
Directories

14.1 Announcements

- Coming attractions:

Event	Subject	Due Date			Notes
asgn3	hencode/hdecode	Fri	Nov 3	23:59	
lab05	mypwd	Mon	Nov 6	23:59	
asgn4	mytar	Mon	Nov 27	23:59	
asgn5	mytalk	Fri	Dec 1	23:59	
lab07	forkit	Mon	Dec 4	23:59	
asgn6	shell	Fri	Dec 8	23:59	

  Use your own discretion with respect to timing/due dates.

```
*** (To get that tree, though,
    requires exponential growth. Each subtree would have to have the
    same number of characters.  Doubling each time means the number of
    characters in the file would be on the order of 2^256.  To put
    that in context, the number of particles in the universe is
    estimated to be somewhere between 2^232 and 2^289.  You're
    probably safe.)

    In fact, since our total count is limited to (3^32-)1 * 256, any file
    we can encode can't have more than a 40-bit code.
```

- valgrind is your friend. Do not ignore it.
- Consider turning on the optimizer. It turns on the dataflow engine and can find errors.
- Garbage collected or not
- Assignment stats:

Assignment	Submitted	Make	Compiled	Passed all	Passed none	complaints
Lab01	64	—	63	63	0	0(!)
Asgn1	63	—	54	50	0	0(!)
Lab02	63	—	62	17	3	0(!)
Lab03	57	—	54	15	10	0(!)

  I published the test suites…

- documentation? Any?
- Test harness? Run it!
- Comment!
- Don't cheat

14.2 Review: Unbuffered IO

```
int open(const char *pathname, int flags, mode_t mode);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
int close(int fd);
off_t lseek(int fildes, off_t offset, int whence);
```

14.3 Onwards: lseek(2)

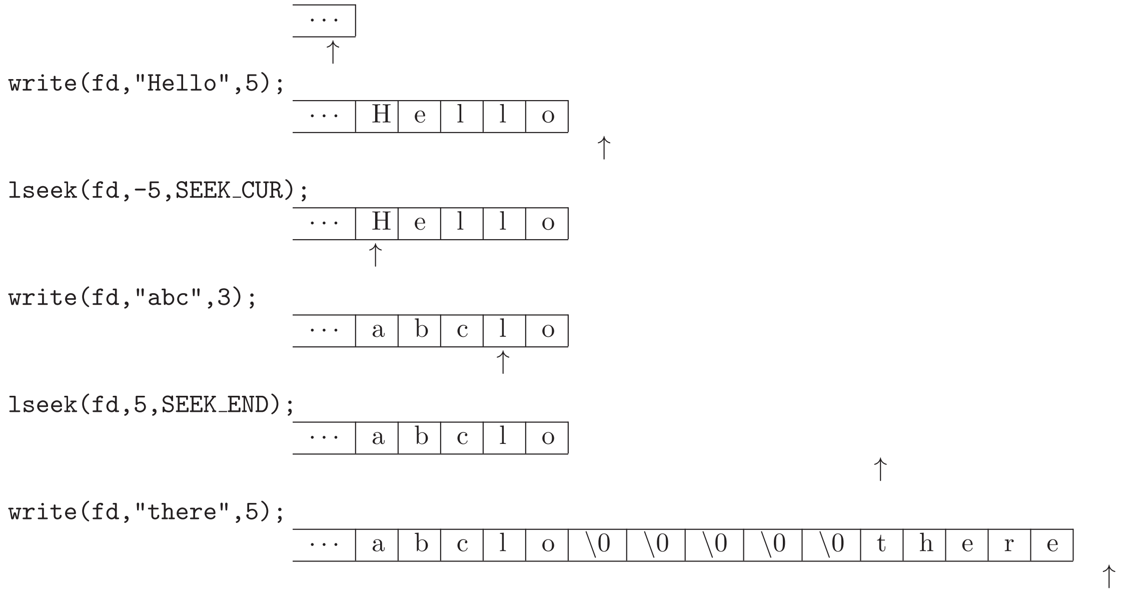
Move the file pointer (or find out where it is). This movement has no effect on the file until the next write.

`lseek(2)` introduces the concept of "holes", places in the file where nothing has ever been written. These locations are read as `'\0'`.

```
SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

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

DESCRIPTION
       The  lseek  function  repositions  the  offset of the file
       descriptor fildes to the argument offset according to  the
       directive whence as follows:

       SEEK_SET
              The offset is set to offset bytes.

       SEEK_CUR
              The offset is set to its current location plus off-
              set bytes.

       SEEK_END
              The offset is set to the size of the file plus off-
              set bytes.
```

Examples:

`lseek(fd, 0, SEEK_SET)`	rewind the file
`lseek(fd, 0, SEEK_END)`	get ready to append
`pos = lseek(fd, 0, SEEK_CUR)`	get current location



```
write(fd,"Hello",5);
lseek(fd,-5,SEEK_CUR);
write(fd,"abc",3);
lseek(fd,5,SEEK_END);
write(fd,"there",5);
```

14.3.1 Structure of the filesystem

The filesystem consists of directories containing links to files and other directories. Each directory has a self-link called "." and a parent link called "..". The parent of "/" is "/".[3]

- The filesystem is a collection of files and directories
- *The Filesystem* may in fact be a collection of smaller mounted filesystems.
- All file properties (of interest) are described by the file's i-node. (even if it's not actually implemented that way)
- Directories are files that know where other files are stored:
  - Direct reading and writing of directories is restricted to the kernel.
  - All directories have at least two entries, "." and ".."
  - Directory contents consist of links.
- Files are connected to the system via links:
  - connects a name to an i-node
  - changing a link only requires access to the directory, not the file.
  - files are only removed when the last link is removed

[3] That is, dot dot of slash is slash :)

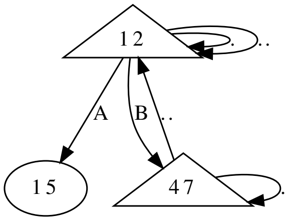
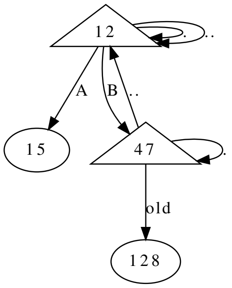
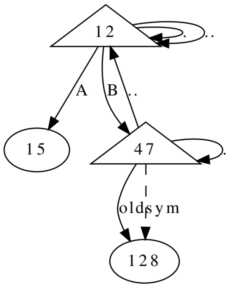
- hard links can only be made within a filesystem.
- only root may make a hard link to a directory

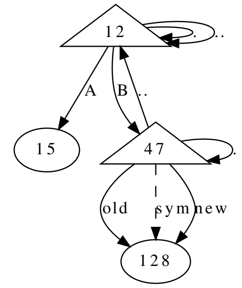
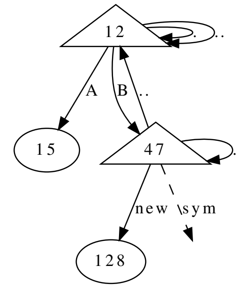
- Symbolic links:
  - like a link, but links to a name, not an i-node.
  - More fragile than a hard link, but can work across filesystems and to directories.

14.3.2 From last time: Links, symbolic and otherwise

A rather detailed discussion of adding/removing elements from directory trees ensued here.

Consider the process below of creating a file, creating a link to it, creating a symlink, then unlinking the original. Assume the directory at inode 12 is the root of the filesystem.

<table>
<tr><th>Command</th><th>/
(inode 12)</th><th>/B
(inode 47)</th><th>Resulting Tree Structure</th></tr>
<tr>
<td>—</td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>12</td></tr><tr><td>..</td><td>12</td></tr><tr><td>B</td><td>47</td></tr><tr><td>A</td><td>15</td></tr></table></td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>47</td></tr><tr><td>..</td><td>12</td></tr></table></td>
<td>
</td>
</tr>
<tr>
<td><code>cd /B</code>
<code>ls &gt; old</code></td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>12</td></tr><tr><td>..</td><td>12</td></tr><tr><td>B</td><td>47</td></tr><tr><td>A</td><td>15</td></tr></table></td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>47</td></tr><tr><td>..</td><td>12</td></tr><tr><td>old</td><td>128</td></tr></table></td>
<td>
</td>
</tr>
<tr>
<td><code>ln -s old sym</code></td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>12</td></tr><tr><td>..</td><td>12</td></tr><tr><td>B</td><td>47</td></tr><tr><td>A</td><td>15</td></tr></table></td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>47</td></tr><tr><td>..</td><td>12</td></tr><tr><td>old</td><td>128</td></tr><tr><td>sym</td><td>“old”</td></tr></table></td>
<td>
</td>
</tr>
</table>

<table>
<tr><th>Command</th><th>/
(inode 12)</th><th>/B
(inode 47)</th><th>Resulting Tree
Structure</th></tr>
<tr>
<td><code>ln old new</code></td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>12</td></tr><tr><td>..</td><td>12</td></tr><tr><td>B</td><td>47</td></tr><tr><td>A</td><td>15</td></tr></table></td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>47</td></tr><tr><td>..</td><td>12</td></tr><tr><td>old</td><td>128</td></tr><tr><td>sym</td><td>“old”</td></tr><tr><td>new</td><td>128</td></tr></table></td>
<td>
</td>
</tr>
<tr>
<td><code>unlink old</code></td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>12</td></tr><tr><td>..</td><td>12</td></tr><tr><td>B</td><td>47</td></tr><tr><td>A</td><td>15</td></tr></table></td>
<td><table><tr><th>Name</th><th>i-node</th></tr><tr><td>.</td><td>47</td></tr><tr><td>..</td><td>12</td></tr><tr><td>—</td><td>—</td></tr><tr><td>symlink</td><td>“old”</td></tr><tr><td>new</td><td>128</td></tr></table></td>
<td>
</td>
</tr>
</table>

14.4 Manipulating the filesystem

```
#include <unistd.h>

int link(const char *oldpath, const char *newpath);
int unlink(const char *pathname);

int symlink(const char *oldpath, const char *newpath);
int readlink(const char *path, char *buf, size_t bufsiz);
         returns character count or -1 (e.g., ENAMETOOLONG)
         (not nul-terminated)

int mkdir(const char *pathname, mode_t mode);
int rmdir(const char *pathname);

int chdir(const char *path);
int fchdir(int fd);

char *getcwd(char *buf, size_t size);
         NULL and ERANGE on error
```

14.5 Directories

Note, that even though there are many fields described in the `struct dirent`, only `d_ino` and `d_name` are required by POSIX.1 and its extension. (POSIX.1 only requires `d_name`.)

```
#include <dirent.h>

       struct dirent
       {
           long d_ino;                 /* inode number */
           off_t d_off;                /* offset to this dirent */
           unsigned short d_reclen;    /* length of this d_name */
           char d_name [NAME_MAX+1];   /* file name (null-terminated) */
       }


DIR *opendir(const char *name);
struct dirent *readdir(DIR *dir);
off_t telldir(DIR *dir);
void rewinddir(DIR *dir);
void seekdir(DIR *dir, off_t offset);
int closedir(DIR *dir);
```