CS 33

Files Part 2
Directories

unix  etc  home  pro  dev

passwd  motd

twd

unix ... 

slide1  slide2
## Directory Representation

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Inode Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory entry</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>..</td>
<td>1</td>
</tr>
<tr>
<td>unix</td>
<td>117</td>
</tr>
<tr>
<td>etc</td>
<td>4</td>
</tr>
<tr>
<td>home</td>
<td>18</td>
</tr>
<tr>
<td>pro</td>
<td>36</td>
</tr>
<tr>
<td>dev</td>
<td>93</td>
</tr>
</tbody>
</table>
$ ln /unix /etc/image
# link system call
## Directory Representation

<table>
<thead>
<tr>
<th>Directory</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>..</td>
<td>1</td>
</tr>
<tr>
<td>unix</td>
<td>117</td>
</tr>
<tr>
<td>etc</td>
<td>4</td>
</tr>
<tr>
<td>home</td>
<td>18</td>
</tr>
<tr>
<td>pro</td>
<td>36</td>
</tr>
<tr>
<td>dev</td>
<td>93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Directory</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>4</td>
</tr>
<tr>
<td>..</td>
<td>1</td>
</tr>
<tr>
<td>image</td>
<td>117</td>
</tr>
<tr>
<td>motd</td>
<td>33</td>
</tr>
</tbody>
</table>
Symbolic Links

% ln -s /unix /home/twd/mylink
% ln -s /home/twd /etc/twd
# symlink system call
Working Directory

- Maintained in kernel for each process
  - paths not starting from “/” start with the working directory
  - changed by use of the `chdir` system call
    » `cd` shell command
  - displayed (via shell) using “pwd”
    » how is this done?
Symbolic Links

Quiz 1
What is the working directory after doing
\texttt{cd /home/twd/mylink/..}?
\begin{itemize}
  \item a) /
  \item b) /unix
  \item c) /home/twd
  \item d) /home/twd/mylink
\end{itemize}
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int options [, mode_t mode])

- options
  » O_RDONLY    open for reading only
  » O_WRONLY    open for writing only
  » O_RDWR      open for reading and writing
  » O_APPEND    set the file offset to end of file prior to each write
  » O_CREAT     if the file does not exist, then create it, setting its mode to mode adjusted by umask
  » O_EXCL      if O_EXCL and O_CREAT are set, then open fails if the file exists
  » O_TRUNC     delete any previous contents of the file
  » O_NONBLOCK  don’t wait if I/O can’t be done immediately
File Access Permissions

• Who’s allowed to do what?
  – who
    » user (owner)
    » group
    » others (rest of the world)
  – what
    » read
    » write
    » execute
Permissions Example

$ ls -lR
.
 total 2
 drwxr-x---x  2 tom adm  1024 Dec 17 13:34 A
 drwxr-------- 2 tom adm  1024 Dec 17 13:34 B

./A:
 total 1
 -rw-rw-rw-  1 tom adm  593 Dec 17 13:34 x

./B:
 total 2
 -r--rw-rw-  1 tom adm  446 Dec 17 13:34 x
 -rw-----rw-  1 trina adm  446 Dec 17 13:45 y
Setting File Permissions

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

– sets the file permissions of the given file to those specified in `mode`

– only the owner of a file and the superuser may change its permissions

– nine combinable possibilities for `mode` (read/write/execute for user, group, and others)

  » S_IRUSR (0400), S_IWUSR (0200), S_IXUSR (0100)
  » S_IRGRP (040), S_IWGRP (020), S_IXGRP (010)
  » S_IROTH (04), S_IWOTH (02), S_IXOTH (01)
Umask

- Standard programs create files with “maximum needed permissions” as mode
  - compilers: 0777
  - editors: 0666
- Per-process parameter, `umask`, used to turn off undesired permission bits
  - e.g., turn off all permissions for others, write permission for group: set `umask` to 027
    » compilers: permissions = 0777 & ~(027) = 0750
    » editors: permissions = 0666 & ~(027) = 0640
  - set with `umask` system call or (usually) shell command
Creating a File

• Use either `open` or `creat`
  - `open(const char *pathname, int flags, mode_t mode)`
    » flags must include O_CREAT
  - `creat(const char *pathname, mode_t mode)`
    » open is preferred
• The `mode` parameter helps specify the permissions of the newly created file
  – permissions = mode & ~umask
Link and Reference Counts

```c
int fd = open("file", O_RDONLY);
    // file’s reference count
    // incremented by 1
unlink("file");
    // file’s link count
    // decremented by 1
close(fd);
    // file’s reference count
    // decremented by 1
```

```
link count == 2
```

**unix**  **etc**  **home**

**image**  **twd**
int main() {
    int fd = creat("file", 0666);
    unlink("file");
    PutStuffInFile(fd);
    ReadStuffFromFile(fd);
    return 0;
}

Assume that *PutStuffInFile* writes to the given file, and *ReadStuffFromFile* reads from the file.

a) This program is doomed to failure, since the file is deleted before it’s used
b) Because the file is used after the unlink call, it won’t be deleted
c) The file will be deleted when the program terminates
Interprocess Communication (IPC)
Interprocess Communication: Same Machine I
Interprocess Communication: Same Machine II

Shared Memory
Interprocess Communication: Different Machines

Diagram showing two machines connected through the internet.
Intramachine IPC

$cslab2e who | wc -l
Intramachine IPC

```c
int fd[2];
pipe(fd);
if (fork() == 0) {
    close(fd[0]);
    close(1);
    dup(fd[1]); close(fd[1]);
    execlp("who", "who", 0); // who sends output to pipe
}
if (fork() == 0) {
    close(fd[1]);
    close(0);
    dup(fd[0]); close(fd[0]);
    execlp("wc", "wc", ":-1", 0); // wc gets input from pipe
}
close(fd[1]); close(fd[0]);
// ...$
cslab2e who | wc -l
```
Pipes

• Pro
  – really easy to use
  – anonymous: no names to worry about

• Con
  – anonymous: can’t give them names
    » communicating processes must be related
Named Pipes

```

mkfifo("/u/twd/service", 0622);
    // creates a named pipe (FIFO) that
    // anyone may write to but only whose
    // owner may read from

int wfd = open("/u/twd/service", O_WRONLY);
write(wfd, request, sizeof(request));
    // send request in one process

int rfd = open("/u/twd/service", O_RDONLY);
read(rfd, request, sizeof(request));
    // receive request in another process
```
Client/Server

Client

server

request pipe

response pipe
Intermachine Communication

• Can pipes and named pipes be made to work across multiple machines?
  – covered soon ...
  » what happens when you type
     who | ssh cslab3a wc -l
     ?