I. What is systems programming?

1. The naive, simple model of programming
Typical program in this model

```c
main()
{
    int c;
    while ( ( c=getchar() ) != EOF )
        putchar(c);
}
```

Reality
Reality

- Lots of users
- Lots of programs
- Lots of disk files
- Lots of devices

Somehow, these are all connected

How are they all connected?

This looks too complicated
The role of the operating system

To manage all the resources and to connect the various devices to the correct programs.

Providing Services to Program

Programs request services from the O/S. The O/S contains code to provide services.
We shall learn about the services provided by the kernel

- Input/Output
- Process Management
- Memory
- Devices: tapes, CDs, mice...
- Timers
- IPC
- Network

Our method for understanding Unix system services:

- Seeing how they are used
- Learning about the system calls to invoke the services
- Writing our own versions of various system programs
Unix from the User's Perspective

- login, run programs, logout
- directories and files
- programs
- I/O control
- pipes
- online documentation
Files and Directories

- `ls`
- `cd`
- `pwd`
- `mkdir`
- `rmdir`

Operations on Files

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>Unix Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>examine a file</td>
<td><code>cat</code>, <code>more</code>, <code>less</code></td>
</tr>
<tr>
<td>copy a file</td>
<td><code>cp old new</code></td>
</tr>
<tr>
<td>delete a file</td>
<td><code>rm</code></td>
</tr>
<tr>
<td>rename a file</td>
<td><code>mv old new</code></td>
</tr>
<tr>
<td>print a file</td>
<td><code>lpr</code></td>
</tr>
</tbody>
</table>
File permissions

Every file has an "owner" and three groups of file permission attributes

- rwx rwx rwx
  owner  group  world

  r = read,  w=write,  x=execute

Using the 3-Step Method: more

Step 1: What does it do?
Try it and read the manpage

purpose:
  more shows a file page by page
usage:
  more filename
  or
  more < filename
  or
  command | more
more: the logic

Step 2: How does it work?
  Looks pretty simple

Action:

- open file
- show 24 lines
- print [More X%?]
- read user input (space or Enter or 'q')
  - if Enter, advance one line
  - if space
  - if 'q', exit

How difficult was that?

Unix programming is not as difficult as you think it is, but it's not as easy as you first imagine...

Let's code our version of more and test it
Testing more: simple case

case 1:

more filename

Result: Works ok

Testing more: pipeline case

case 2:

who | more

Result: trouble...
Understanding more

The 'real' version of more has to read from the keyboard even when standard input is attached to a pipe. Here's how it works:

![Diagram showing standard input and /dev/tty connections]

Solution: read user input from /dev/tty.

More questions

Reading from /dev/tty solves the pipeline problem, but...

- The real more does not echo the 'q' or the space
- The real more does not wait for the Enter key
- The real more knows how many lines are on the terminal

How does more control the input device?
Every device has a collection of settings that determine how the device operates. A program can read and modify device settings. For example, more can turn off character echoing.

Yet **more** questions

What about the percentage of the file more shows?

*How does it know that?*

What about the reverse video?

*How does it do that?*

How about pressing 'v' to exit the file?

*How does that work?*

less can go backwards?

*Does the program read backwards?*
Learning from Unix Programs

Common Unix programs use kernel services. We can learn about those services and how they are used by studying and writing these common programs.