

## Intro to Unix

- Text-based interface.
- Type commands with arguments at a "prompt":

- You can use this as an adjunct to (rather than replacement of) the **GUI** interface.
- The "prompt" may be anything, not necessarily "\$". It doesn't matter what the prompt is.
- Note: In notes I often use the convention "\$" to indicate the prompt, no matter what the prompt is.
- Can launch GUI programs:

## PATH

- ```
echo $PATH
```

- ## Hierarchical Directory Structure

- `cd ..` Go to parent directory  
e.g `/users/staff`

- `.` Current directory
- `/` root directory
- `$HOME` home directory
- `$HOME/public_html` public web space
- `/tmp` system temporary files

- Hierarchical file system - /directory/sub-directory/file
- Forward slash (this is why Web is forward slash).

## CA170: Week 2

### Intro to Unix

#### Terminal

- Command-line keys:
  - Ctrl-C - interrupt program
  - Ctrl-A - Home
  - Ctrl-E - End
  - Command line shortcuts
  - clear - clear screen
- ls
  - ls
  - ls -l
  - echo \* (all normal, non-hidden files and dirs)
  - ls -l \* (if dir, will display contents)
  - ls -ld \* (if dir, display name not contents)
- hidden files
  - ls -a
  - ls -al
  - echo .\* (all hidden files)
  - echo .\* \* (all files)
  - ls -l .\* (don't need -a if going to actually name the hidden files)
  - ls -ld .\*
- environment variables
  - set
  - set > file.txt
  - env
  - env | sort
  - echo HOME - the string "HOME"
  - echo \$HOME - the global (environment) variable HOME
  - PATH = list of dirs searched for command when you type a command, separated by colons (:)
  - SHELL = shell we are using (default here is bash)
- Shared, multi-user, file system:
  - HOME = something like:  
/users/yourgroup/yourusername
  - My test account is here:  
/users/tutors/mhtest15
- current and parent dirs
  - . (current dir)
  - Example: When was the last change to the current dir:  
ls -ld .
  - .. (parent dir)
- cd
  - go home: cd
  - go up to parent dir: cd ..
  - go back to last dir: cd -

- command history
  - up/down arrows
  - history
  - h (alias)
  - !n - repeat command n
  - !string - repeat last command that began with string
  - !c - repeat last command that began with character c
  - !! - last command

## Disk Quota

- You have 500 M disk quota.
- quota
- quota -s
- If you get "Disk quota exceeded"
  - You need to delete some files.
  - To see which dirs are taking up room:
  - `du | sort -n`
- MetaTracker uses disk space
  - A program called MetaTracker may be using a lot of your disk space.
  - MetaTracker uses a directory \$HOME/.cache
  - This is safe to delete:
    - `cd`
    - `rm -r .cache`
  - See now instruction in default .bashrc that removes .cache for each new shell.
  - If .cache will not delete because MetaTracker is running:
    - Use ps to find "tracker" processes and kill them.
    - Then remove .cache
- Wine uses disk space
  - Another program that uses a lot of disk space is Wine.
  - `cd`
  - `rm -r .wine`

## Shakespeare

- For testing manipulation of data with Linux commands, it helps if we have some data to manipulate.
- So I have installed the works of Shakespeare to use for testing.
- I downloaded the works of Shakespeare from the free eBooks site at the University of Adelaide.
- This has now closed. There are old copies in the Internet Archive:
  - Shakespeare page
  - ZIP file
- I have a copy on the shared file system here:
  - `/users/tutors/mhtest15/share/shakespeare`
- You can "cd" to this directory.

- How to view these local web pages:
  - These web pages are not on the web.
  - You can still view these local files through a web browser, using this URL:
  - `file:///users/tutors/mhtest15/share/shakespeare/home.html`
  - You will have to paste the URL into your browser address bar. Modern browsers do not allow links from `http://` to `file://` for security reasons.
- Once you are in `file://` mode, you can follow links from `file://` to `file://`
- Or cd into my directory, and:
 

```
firefox home.html &
```
- I am sharing these through the shared file system, not through http.
- Permissions need to be something like:
  - `drwx--x--x`      `/users/tutors/mhtest15`
  - `drwxr-xr-x`      `/users/tutors/mhtest15/share`

## CA170: Week 3

### Intro to Unix

#### Files

|             |                                                                                              |
|-------------|----------------------------------------------------------------------------------------------|
| ls          | List files                                                                                   |
| ls -a       | Show "hidden" files (begin with ".")                                                         |
| ls -l       | Detailed                                                                                     |
| ls -alR     | Recursive                                                                                    |
| cat (file)  | Type file out in command-line window                                                         |
| more (file) | Type file, pause for each screenful<br>enter for new line, space for next page, q to<br>quit |

#### Manipulating Files

- Human can use GUI file manager. Programs can do anything a human can do, with these commands.
- Human sometimes types these commands too.

|       |                     |
|-------|---------------------|
| cp    | Copy files          |
| mv    | Move / Rename files |
| rm    | Remove files        |
| mkdir | Make directory      |
| rmdir | Remove directory    |

- Notes have tutorials

#### Detach Program (&)

|          |                                                                |
|----------|----------------------------------------------------------------|
| (prog) & | Launch a process detached<br>from command-line (e.g. windowed) |
| (prog)   | Command-line frozen until prog exits.                          |

eog file.jpg &      Launch image viewer **Eye of GNOME** on file

#### Web Browser

firefox &      Launch Web browser from command-line  
firefox "URL" &

google-chrome &

**konqueror** "URL" &

- Firefox multiple windows is confusing on Gnome.
- To find other windows: "Activities"
- Or Alt-Tab and pause on Firefox and they pop up.

## sort

**sort**                Sort alphabetically (pipe some stream into sort)  
sort -n               Sort numerically  
sort -r               Reverse sort

sort by 5th column:  
sort -k 5

old syntax to sort 5th column:  
sort +4

## grep

grep                Search for a string in a file or files

grep (string) (file)  
grep (string) \*html

grep -i (string) (file)   Ignore case  
grep -v (string) (file)   Return lines that do NOT match

## find

find                Find files by type/date/name, in this dir or below

find -type d           Find all dirs  
find -mtime -1        Find files modified today

## du

- Links to more notes in notes

du                        Space used by me

du | sort -n              # why -n?

default is k

listing in M  
du -BM

listing in G  
du -BG

top-level only  
--max-depth=1

## Printing

lp (file)                Print  
lpr (file)               Print (on some systems)  
lp -Pl128 (file)        Print on printer l128 (L128)

lpq                      See print queue  
lprm                     Remove job from queue

## cal

|             |                               |
|-------------|-------------------------------|
| <b>cal</b>  | Calendar                      |
| cal 8 1752  | Calendar for Aug 1752         |
| cal 9 1752  | Calendar for <b>Sept 1752</b> |
| cal 10 1752 | Calendar for Oct 1752         |

## which

which (prog)  
what runs if "prog" is typed  
may return nothing if prog is an **alias**

which ls

type (prog)  
returns path of prog  
or else shows what prog is alias for

type h  
type history

whereis (prog)                      Where the binary, source, manual pages are for this prog  
whereis perl

## Misc

|       |                         |
|-------|-------------------------|
| df -h | Show space on all disks |
| df -k | exact kilobytes         |

w                                      Who is logged in  
(see this when you ssh student.computing.dcu.ie)

(command) ; (command)      Multiple commands on same line

## wget

- **Command line HTTP client**

|                  |                                             |
|------------------|---------------------------------------------|
| wget -q -O - URL | Download URL, quiet, output to command-line |
|------------------|---------------------------------------------|

|                             |                |
|-----------------------------|----------------|
| wget -q -O - URL > file.htm | Output to file |
|-----------------------------|----------------|

|                         |                |
|-------------------------|----------------|
| wget -q -O file.htm URL | Output to file |
|-------------------------|----------------|

|                                                                                          |                     |
|------------------------------------------------------------------------------------------|---------------------|
| wget -q -O - http://site/file.jpg > file.jpg<br>(output JPEG to command-line won't work) | Output JPEG to file |
|------------------------------------------------------------------------------------------|---------------------|

|                                          |                     |
|------------------------------------------|---------------------|
| wget -q -O file.jpg http://site/file.jpg | Output JPEG to file |
|------------------------------------------|---------------------|

- **Link to more in notes**

## Absolute and Relative Paths

- Relative path of a file

`index.html`

- What file that refers to depends on what directory you are in now.
- It looks for `index.html` in the current directory.

`../index.html`

- is also relative path. It looks for `index.html` in the parent of the current directory.

| Directory before        | Command                   | Directory after                   |
|-------------------------|---------------------------|-----------------------------------|
| <code>/users/gdf</code> | <code>cd users/ec2</code> | <code>/users/gdf/users/ec2</code> |
| <code>/users/gdf</code> | <code>cd ../ec2</code>    | <code>/users/ec2</code>           |
| <code>/users/gdf</code> | <code>cd ec2</code>       | <code>/users/gdf/ec2</code>       |

- Absolute path of a file

`/dir/dir/dir/public_html/index.html`

- Gives the full "path" from the root down to the file.
- Refers to the same file no matter what directory you are in.

| Directory before        | Command                    | Directory after         |
|-------------------------|----------------------------|-------------------------|
| <code>/users/gdf</code> | <code>cd /users/ec2</code> | <code>/users/ec2</code> |

## Case Sensitivity

- Case matters in filenames in UNIX (this is why case often matters on Web).
- Question: Is case sensitivity a good thing? Or is it a flaw in UNIX?
- Advantages of case sensitivity:
  - More readable code. You know what to expect.
  - More variables. `num` and `NUM` and `Num`
  - Set up conventions, so that `NUM` probably refers to a compile-time-coded constant, `num` is a real-time-changing variable, etc.
  - Quicker/simpler searches on strings and changes of strings, since can just search for the literal string.
  - Better to be case-sensitive for passwords. - Larger space to pick from. Harder to guess. Good to be "unforgiving" for security.
- Not much return for such huge disadvantages:
  - Millions of programmer and user hours lost on case not right.
  - Millions of failed "404 Not Found" website hits because of wrong case in the URL.
- Solutions to "404 Not Found" because of case:
  - Set up program to handle 404. See My "404 Not Found" Handler
  - Detach website URLs from underlying (case sensitive) file system.
    - e.g. Content Management System.



## Filenames and Special Characters

- Long file names and multiple periods OK.
- e.g. product.4652.suppliers.us.html
- UNIX had long filenames from the start, unlike DOS/Windows with the 8.3 format (still part of legacy command-line interaction on Windows).
- openSUSE uses ext3 file system (and here)
- Comparison of file systems
- Limits
  - ext3 allows 255 char file names
  - Linux has a maximum path name limit of 4096 chars.
  - Part of the source code on my install:  

```
/usr/include/linux/limits.h
```
  - shows the following:  

```
#define NAME_MAX      255      /* # chars in a file name */
#define PATH_MAX      4096     /* # chars in a path name
including nul */
```
- Avoid these
  - If the command-line is used to address files, it is best to avoid many special characters in filenames.
  - Avoid these chars in filenames, because they have meaning to the Unix command-line and utilities:  

```
space (separate arguments)
# comment
< redirection
> redirection
` result of a program
| pipe
& detach process
; separate multiple commands on the same line

* wildcard
? wildcard
^ start of line
$ end of line / variable value
[ pattern matching
] pattern matching
\ "quoted" character
/ should be in pathname, not filename
' string delimiter
" string delimiter
! shell history
```
  - On many UNIX/Linux distributions (e.g. openSUSE) you can actually put these chars in filenames. But the file may then be hard to work with at the command-line, and naive scripts may crash.
  - If you just point-and-click your UNIX files (which is allowed too) then you can use many of these characters.

- Stick to these

- If you're going to use the command-line, best to just use these chars in filenames:

0-9

a-z

A-Z

Use these inside filename only, not at start or end:

.

-

\_

- Filename (notes on wiki)
- File name characters (notes in wiki)
- UNIX / Win are quite restrictive - lot of bad chars.
- Mac can use almost any char.
- Though could then get problems on Mac command line.
- Can get problem on UNIX when you download or copy in files from another OS with odd chars in file names.
- Filenames with spaces are creeping into the UNIX world, but old scripts may fail with them.
- Find any file or directory below current dir with spaces in its pathname:

```
$ find . | grep ' '
```

## Notes on File Protection

### File Protection

- "ls -l" shows something like:

```
-rwxr-xr-- 1 userid groupid 153 Nov  6 2008 filename
```

```
-      file (d for directory, l for link/shortcut)
rwx   User (u) can read,write,execute.
r-x   Other members of group (g) can read,execute only.
r--   Other people (o) can read only.
```

set via the "chmod" command.

see **"man chmod"**

```
      user      group      other
[ ][ ][ ]  [ ][ ][ ]  [ ][ ][ ]
```

```
r - read
w - write
x - execute
```

- e.g. user can do everything, group/others can do nothing:

```
chmod u+rwx,go-rwx file
```

- result:

```
-rwx----- 1 userid groupid 153 Nov 6 2008 filename
```

- There is also a number that corresponds to each permission setting:
  - chmod converter (and search for more) - more in notes
- Default permissions for new files: umask

### User bits

- Note if turned off, user has power to turn them on any time.
- So these can only be for some kind of temporary self-check.

|             |                                                                                                                                              |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| [r] [w] [-] | Don't execute by accident.<br>Because UNIX will try to execute any text file as shell script if name is typed.<br>e.g. text files, web pages |
| [r] [-] [x] | write-protect for safety<br>annoying?                                                                                                        |
| [r] [-] [-] | both of above                                                                                                                                |
| [r] [w] [x] | normal                                                                                                                                       |

### group/other bits

|                            |                             |
|----------------------------|-----------------------------|
| [r] [w] [x]<br>[r] [w] [-] | Shared writable file        |
| [r] [-] [x]<br>[r] [-] [-] | Shared read-only file       |
| [-] [-] [-]                | Normal - Hidden from others |

## Minimum Needed for Web Files

- (Web server is "other".)

Web pages need r:

`-rwx---r--`

PHP scripts only need r, not x:

`-rwx---r--`

## Notes on Directory Protections

```
      user      group      other
[ ][ ][ ]  [ ][ ][ ]  [ ][ ][ ]
```

r - read (can do ls)

w - write

x - search (can access files given their name)

### User bits

- Note if turned off, user has power to turn them on any time.

|             |                                       |
|-------------|---------------------------------------|
| [r] [-] [x] | write-protect for safety<br>annoying? |
| [r] [w] [x] | normal                                |

### group/other bits

|             |                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [r] [w] [x] | shared writable directory<br>can create/delete files                                                                                                                                                                                                                                                                                                                                                       |
| [r] [-] [x] | shared read-only directory<br>can do ls                                                                                                                                                                                                                                                                                                                                                                    |
| [-] [-] [x] | shared read-only dir<br>can't do ls<br>can access file if know its name<br>can't explore without filenames<br><br>Example: "share" in my home dir.<br>You just need to know this dir exists.<br><br>Example: web dir<br>Can only browse named files. The names are <i>in the links</i> . The site advertises a starting point (a web page from which all other web pages can be found by following links). |
| [-] [-] [-] | normal - hidden                                                                                                                                                                                                                                                                                                                                                                                            |

### Raw listing of files on web servers

- It used to be that we could demo the difference between r and x for web directories.
- In my web dir:

```
drwx---r-x    readabledir
drwx-----x    executabledir
```
- readabledir/file.html - link in notes
- executabledir/file.html - link in notes
- executabledir - index.html does not exist, so it just returns error. - link in notes
- readabledir - index.html does not exist, but dir is readable, so what it used to do is return a raw listing of files. - link in notes

### Raw listing of files is now turned off

- The above (raw listing of files) does not work any more on student.computing.dcu.ie.
- On Apache, the behaviour of listing directory contents or not can be controlled with `Options +Indexes` (or `Options -Indexes`) in `.htaccess` files.
- It is now turned off.

### Minimum needed for web directories

- (Web server is "other".)

`drwx-----x`

## CA170: Week 4

### More Unix

#### Pipes and redirections

- Very powerful features.
- Pipes - Send output of one program to input of other.
  - e.g. Search for all lines in the file that contain "DCU" in any case, except those containing "Computing" in any case, and sort the remaining lines:

```
cat file | grep -i dcu | grep -iv computing | sort
```

- Backquotes - Capture the output of a program as a string:
- `echo Current directory is `pwd` and date is `date``
- Redirection - Read input from a file, send output to a file.

```
prog < inputfile > outputfile
```

```
cat file | grep -i student > studentlist
```

```
cat file | grep -i staff > stafflist
```

- All these are the same:

```
grep string file
```

```
grep string < file
```

```
cat file | grep string
```

- Append:

```
cat morestuff >> studentlist
```

- Unix pipeline - link in notes

#### Filename completion

- Start typing filename, hit special key to complete it.
- Linux - Tab
  - If typing command - will search whole PATH for matches
  - If just listing a file - will only search current dir
  - If more than one choice - hit Tab twice to show
- Solaris - Esc

#### Processes

|                                |                                  |
|--------------------------------|----------------------------------|
| <code>ps</code>                | See what processes are running   |
| <code>kill (process id)</code> | Terminate some of your processes |
| <code>kill -KILL (pid)</code>  | Definite kill                    |
| <code>kill -1</code>           | Kill all my processes            |

`PPID`                      parent process of this process

- `ps (see man)` - link in notes
- `kill (see man)` - link in notes

`xkill &`                      Kill the next thing I click on

`nice`                      Run something at low priority deliberately

`time`                      Time a run of some program

- `xkill (see man)` - link in notes
- `killall (see man)` - link in notes

- Display of processes
  - Linux command line
    - top (see man) - link in notes
  - Linux GUI:
    - Computer - System Monitor
  - Windows: Task Manager
  - Explanation of some of the data returned: - links to these in notes
    - paging and swapping
    - Threads
    - GDI objects (Windows only)
  -

## ps in DCU

### 1. Normal login in DCU labs:

- You each have your own CPU and memory, sharing a central filesystem.
- "ps" will show that the only processes running on the machine are yours and the Operating System's.
- `ps -Tf`
  - T associated with this terminal
  - f full details
- `ps -u $USER -f`
  - u \$USER associated with this user
- `ps -u $USER -o user,pid,ppid,comm,args`
  - o Show these fields

### 2. If doing ssh to student.computing.dcu.ie:

- Possibly shared CPU with other users.
- student.computing.dcu.ie is a Linux cluster, so some users have their own CPU, others are on shared CPUs, by chance. - link in notes
- To see other people's processes:
 

```
ps -Af
```

-A all processes

## Interrupts

- Usage seems to vary on different variants of UNIX and Linux. You may get something like:

|               |                |
|---------------|----------------|
| Ctrl-S        | Pause          |
| <b>Ctrl-C</b> | Interrupt      |
| <b>Ctrl-D</b> | Kill, Logout   |
| <b>Ctrl-Z</b> | EOF            |
| q             | exit man, more |



## Command line philosophy

- A computer for programmers.
- Unix philosophy
- Provide lots of tools. String together tools with Shell logic, pipes, redirection.
- If tools cannot do it alone, you can assemble a program to do it. "Program" may be just one line long.
- all lower case - fast typing, don't have to hit Shift key
- short command names - fast typing
- Silence, "low-noise environment".
- `rm` all my files, and it just does it. Doesn't even display a message saying they have been removed.
- Note that backquotes would be useless if all programs displayed lots of informational messages as they executed.
- Often an explicit `prog -v "verbose"` option if you want to see informational messages as it executes. But this is normally not the default.

## GUI Philosophy

- A computer for non-programmers.
- (Or for a thing you want to do now for which a command-based approach is not appropriate.)
- If it is not in the pre-defined tasks and menus, then you can't do it. You cannot assemble a program to do it.
- High-noise. - `rm` all my files - dialog box comes up. Are you sure? OK.
  - This dialog makes no sense in Unix - where a program, not a human user, might be issuing the command to `rm` files.
- Note that user interface people say these dialogs are often ignored.
- From Donald Norman, *The Psychology of Everyday Things*, 1988, Ch.5: - link in notes

```
Human - Delete all my most important files.  
System - Are you sure?  
Human - Yes Yes.  
System - Are you really sure?  
Human - Yes Yes.  
System - All your most important files deleted.  
Human - Oh damnit.
```

- Compare with Unix

```
Human - Delete all my most important files.  
System - (Silence.)  
Human - Oh damnit.
```

## Command line on Mac/Windows

- Windows always had DOS command-line, and still does.
- But for many years it was neglected, not as powerful as UNIX command-line.
- People who liked command lines tended to migrate to UNIX/Linux.
- Recently, though, more powerful command shells have been introduced on Windows.
- And now, Linux shell on Windows.
- Mac for years had no command-line at all.
- But now has UNIX command-line.
- Typical modern Mac has powerful UNIX command-line with `bash`, `csh`

## Notes on File Protection

### File Protection

- "ls -l" shows something like:

```
-rwxr-xr-- 1 userid groupid 153 Nov  6 2008 filename
```

```
-      file (d for directory, l for link/shortcut)
rwx   User (u) can read,write,execute.
r-x   Other members of group (g) can read,execute only.
r--   Other people (o) can read only.
```

set via the "chmod" command.

see **"man chmod"**

```
      user      group      other
[ ][ ][ ]  [ ][ ][ ]  [ ][ ][ ]
```

```
r - read
w - write
x - execute
```

- e.g. user can do everything, group/others can do nothing:

```
chmod u+rwx,go-rwx file
```

- result:

```
-rwx----- 1 userid groupid 153 Nov 6 2008 filename
```

- There is also a number that corresponds to each permission setting:
  - chmod converter (and search for more) - more in notes
- Default permissions for new files: umask

### User bits

- Note if turned off, user has power to turn them on any time.
- So these can only be for some kind of temporary self-check.

|             |                                                                                                                                              |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| [r] [w] [-] | Don't execute by accident.<br>Because UNIX will try to execute any text file as shell script if name is typed.<br>e.g. text files, web pages |
| [r] [-] [x] | write-protect for safety<br>annoying?                                                                                                        |
| [r] [-] [-] | both of above                                                                                                                                |
| [r] [w] [x] | normal                                                                                                                                       |

### group/other bits

|                            |                             |
|----------------------------|-----------------------------|
| [r] [w] [x]<br>[r] [w] [-] | Shared writable file        |
| [r] [-] [x]<br>[r] [-] [-] | Shared read-only file       |
| [-] [-] [-]                | Normal - Hidden from others |

## Minimum Needed for Web Files

- (Web server is "other".)

Web pages need r:

`-rwx---r--`

PHP scripts only need r, not x:

`-rwx---r--`

## Notes on Directory Protections

```
      user      group      other
[ ][ ][ ]  [ ][ ][ ]  [ ][ ][ ]
```

r - read (can do ls)

w - write

x - search (can access files given their name)

### User bits

- Note if turned off, user has power to turn them on any time.

|             |                                       |
|-------------|---------------------------------------|
| [r] [-] [x] | write-protect for safety<br>annoying? |
| [r] [w] [x] | normal                                |

### group/other bits

|             |                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [r] [w] [x] | shared writable directory<br>can create/delete files                                                                                                                                                                                                                                                                                                                                                       |
| [r] [-] [x] | shared read-only directory<br>can do ls                                                                                                                                                                                                                                                                                                                                                                    |
| [-] [-] [x] | shared read-only dir<br>can't do ls<br>can access file if know its name<br>can't explore without filenames<br><br>Example: "share" in my home dir.<br>You just need to know this dir exists.<br><br>Example: web dir<br>Can only browse named files. The names are <i>in the links</i> . The site advertises a starting point (a web page from which all other web pages can be found by following links). |
| [-] [-] [-] | normal - hidden                                                                                                                                                                                                                                                                                                                                                                                            |

### Raw listing of files on web servers

- It used to be that we could demo the difference between r and x for web directories.
- In my web dir:

```
drwx---r-x    readabledir
drwx-----x    executabledir
```
- readabledir/file.html - link in notes
- executabledir/file.html - link in notes
- executabledir - index.html does not exist, so it just returns error. - link in notes
- readabledir - index.html does not exist, but dir is readable, so what it used to do is return a raw listing of files. - link in notes

### Raw listing of files is now turned off

- The above (raw listing of files) does not work any more on student.computing.dcu.ie.
- On Apache, the behaviour of listing directory contents or not can be controlled with `Options +Indexes` (or `Options -Indexes`) in `.htaccess` files.
- It is now turned off.

### Minimum needed for web directories

- (Web server is "other".)  
`drwx-----x`

## CA170: Week 5

### Intro to Shell Programming

#### Intro

- Unix/Linux command-line - One-liner "programs" at the command prompt (e.g. prog piped to prog).
- Shell programming - Text file of multiple Unix/Linux commands (e.g. "if" condition then execute command, else other command).
- The combination of one-liner command-line plus multi-line Shell programs gives you a programmable User Interface, where you can quickly write short programs to automate repetitive tasks.
- Shell program = an interpreted program (i.e. not compiled). Also known as a "Shell script" or "batch file" of UNIX commands.
- Like .BAT files on Windows command line.

#### How to make a Shell Program

1. Put it in a directory that is in the PATH. In this course we will put it in \$HOME/bin
2. It can have any extension (typically no extension).
3. Put valid UNIX commands in it.
4. Make it executable:

```
$ chmod +x file
```

5. Run it by typing its name:

```
$ file
```

```
$ file &
```

#### Alternative ways of working

1. Pass program as arg to shell:

```
$ sh prog
```

- Advantage: Does not have to be executable. Does not have to be in PATH.
- Disadvantage: Have to type "sh" all the time. Have to be in same directory (or else type more complicated path to program).

2. Use file extension:

```
$ prog.sh
```

- Advantage: Can easily see what the file is from a directory listing.
- Disadvantage: Have to type the .sh
- Could be used when no one ever types the name. (e.g. The script is only ever called by a program.)

## Arguments and returns

- A Shell program is different to something typed on the command-line.
- It can have arguments, and can exit at any point.

`$1`      1st command-line argument

`$2`      2nd command-line argument

`...`

`$*`      all arguments

`#`      comment

`exit`      exit the Shell script

`exit 0`      exit with a return code that other progs can query

`$?`      return code of last prog executed

## if, test and flow of control

- Conditional statements - link in notes
- Flow of control statements- link in notes
- Test - link in notes

```
# test if 1st argument = "0"
```

```
if test "$1" = "0"
```

```
then
```

```
    echo "yes"
```

```
else
```

```
    echo "no - first argument is $1"
```

```
fi
```

## Sample Shell Program

### norm - set permissions as open as possible

```
chmod u+rwx,go+rx-w $*
```

### hide - as hidden as possible

```
chmod u+rwx,go-rwx $*
```

### semihide - just open enough as needed for Web

- Directories

```
chmod u+rwx,g-rwx,o+x-rw $*
```

- Files (webpages , images, etc)

```
chmod u+rwx,g-rwx,o+r-xw $*
```

- "norm" could of course replace all 3 if you don't mind granting more access than strictly necessary.

### rmifexists - silent repeated delete

- e.g. Want to be able to repeatedly run:

```
for i in $*
do
    if test -f $i
    then
        rm $i
    fi
done
```

- and not get error message if no \*.bak files found.
- in fact, `rm -f` will do the job with no warnings
- Rm - link in notes

### Recursive rm

- `rm -r` (recursive rm)
- Don't type:
- `rm -rf /`
- "Trying out some Deadly Linux Commands".
- Includes typing `rm -rf /` and other scary commands, including the hilarious: `mv /dev/null`
- Link in notes + link to video



## wipe - clean up editor backup files

```
rmifexists *%
rmifexists .*%
```

```
rmifexists *~
rmifexists .*~
```

```
rmifexists *.bak
rmifexists *.*.bak
```

```
rmifexists *.BAK
rmifexists *.*.BAK
```

1. Easier than having to point-and-click each one. Especially if do this for multiple directories.
2. Safer than typing "rm \*bak" every day. One day you will type "rm \* bak"
3. In general, if you regularly type some command that would be dangerous if you make a typo, it would be better to debug it once and put it in a script and never type it directly again.

## Command-line image processing

- We need libjpeg utilities.
- At DCU this is:
  - Installed on PCs in labs.
  - Not installed on student.computing.dcu.ie (ssh).
- To make 1/4 size versions of 10,000 JPEGs without ever opening an image editor (or doing any work):

```
for i in *jpg
do

    djpeg -scale 1/4 -bmp $i > temp.bmp

    cjpeg temp.bmp > small.$i

done
```

- JPEG needs to be decoded to a BMP (bitmap), then be re-sized, then re-coded back to JPEG.
- Can do this in one line, leaving out the temporary file: Pipe result of djpeg into input of cjpeg.
- Link to BMP in notes

## Extract images from PDFs

- I was once given an archive of thousands of scanned historical images. They were all inside PDFs.
- I automatically extracted all the JPEGs from the PDFs as follows:

```
for i in *pdf
do
    # i = x.pdf

    x=`basename "$i" ".pdf"` # get root filename x (without .pdf
    bit)
```

```
pdfimages -j $i $x # extracts images to x-nnn.jpg
```

done

### Command-line movie processing

- You can do command-line movie processing with ffmpeg.
- Your Shell script can bulk convert, split, join, rotate and resize thousands of videos. Without opening any window.
- ffmpeg command-line examples - [link in notes](#)

## Sample Shell Scripts

### Sample Shell script - filterbaks

- The following is to illustrate that \*. files are only "hidden" by convention.
- We can pipe all our commands through a "filterbaks" program to make lots of other files also hidden by convention:
- Might hide text editor backup files (things like file%, file~, file.bak):
- 

### Filterbaks

- Might make it a separate file, called say "filterbaks":

```
grep -v "%$" | grep -v "~$" | grep -iv "\.bak$"
```

so we can reuse it in directory listing:

```
ls -al $* | filterbaks
```

and in other progs:

```
if test `echo $i | filterbaks`  
then  
...  
fi
```

- Makes backup files invisible everywhere (until needed).
- Finally, it would be more efficient to replace 3 programs piped together with 1 program (with more complex arguments). (Why would this be more efficient?)
- We can do this using the "egrep" program, which can grep for boolean expressions. filterbaks can be rewritten as simply:

```
egrep -iv "%$|~$|\.bak$"
```

- where, inside the string here, | means "OR".

### d - my own ls script

```
ls -l $* | filterbaks
```

## How to debug a program

- With the vast amount of sample code on the Internet, I often see students getting big chunks of sample code and trying vainly to get it to work. I also provide sample code which I ask students to modify. I have noticed students often do not have the right mindset when doing this.

- Here are a few simple tricks for how to debug a program:

1. Strip it down. Remove parts of it.
2. Get smaller parts working first.
3. Don't try to do it all at once.
4. You don't have to delete code. Just comment it out. Then slowly comment it back in.
5. Comment out lines of code:

```
# code
// code
```

6. A trick is to use tabs to make it easy to comment code out and in:

```
#    code
#    code
#    code
#    code
```

7. Comment out blocks of code:

```
/*
code
code
*/
```

8. Insert an exit after reaching a certain stage (comments out everything below it):

```
code
code

    exit

code
Code
```

9. Look at variables half-way through:

```
echo $var
```

```
System.out.print(var);
```

```
console.log(var);
```

10. Build all programs in stages, testing each stage.
11. Slowly comment code back in.
12. Slowly move exit further down or remove it.
13. Slowly remove debug info.

- Debugging is not a mystery. Yes, there are fancy debugging and tracing tools.
- But half the time, a few well-chosen prints, exits and comment-outs are all you need to find the problem.
- Link in notes

## Evolution of a Programmer

### High School / Junior High

```
10 PRINT "HELLO WORLD"
20 END
```

### First Year in College

```
program Hello(input, output)
begin
    writeln('Hello World')
end.
```

### Senior Year in College

```
(defun hello
  (print
   (cons 'Hello (list 'World))))
```

### New professional

```
#include
void main(void)
{
    char *message[] = {"Hello ", "World"};
    int i;

    for(i = 0; i < 2; ++i)
        printf("%s", message[i]);
    printf("\n");
}
```

### Seasoned professional

```
#include
#include
class string
{
private:
    int size;
    char *ptr;
public:
    string() : size(0), ptr(new char('\0')) {}
    string(const string &s) : size(s.size)
    {
        ptr = new char[size + 1];
        strcpy(ptr, s.ptr);
    }
    ~string()
    {
        delete [] ptr;
    }
    friend ostream &operator <<(ostream &, const string &);
    string &operator=(const char *);
};
ostream &operator <<(ostream &stream, const string &s)
{
    return(stream << s.ptr);
}
string &string::operator=(const char *chrs)
{
    if (this != &chrs)
    {
        delete [] ptr;
        size = strlen(chrs);
        ptr = new char[size + 1];
        strcpy(ptr, chrs);
    }
    return(*this);
}
int main()
{
    string str;
    str = "Hello World";
    cout << str << endl;
    return(0);
}
```

### Apprentice Hacker

```
#!/usr/local/bin/perl
$msg="Hello, world.\n";
if ($#ARGV >= 0) {
    while(defined($arg=shift(@ARGV))) {
        $outf = $arg;
        open(FILE, ">" . $outf) || die "Can't write $arg: $!\n";
        print (FILE $msg);
        close(FILE) || die "Can't close $arg: $!\n";
    }
}
else {
    print ($msg);
}
1;
```

### Experienced Hacker

```
#include
#define S "Hello, World\n"
main(){exit(printf(S) == strlen(S) ? 0 : 1);}
```

### Seasoned Hacker

```
% cc -o a.out ~/src/misc/hw/hw.c
% a.out
```

### Guru Hacker

```
% cat
Hello, world.
^D
```

### Junior Manager

```
10 PRINT "HELLO WORLD"
20 END
```

### Middle Manager

```
mail -s "Hello, world." bob@b12
Bob, could you please write me a program that prints
"Hello, world."? I need it by tomorrow.
^D
```

### Senior Manager

```
% zmail jim
I need a "Hello, world." program by this afternoon.
```

### Chief Executive

```
% letter
letter: Command not found.
% mail
To: ^X ^F ^C
% help mail
help: Command not found.
% damn!
!: Event unrecognized
% logout
```

## CA170: Week 6

### More on Shell

#### Pipes and redirection in scripts

- See notes on pipes and redirections on command line too

- Multi-line pipe:

- Shell script can have multi-line pipe. Easier to read. e.g.:

```
cat file |
```

```
grep -i "http:.*dcu.ie" |
```

```
sed -e "s|COMPAPP|computing|g" |
```

```
sed -e "s|compapp|computing|g" |
```

```
sort -u
```

- This example uses sed to do a "search and replace" operation on text.

- Two output streams:

UNIX separates "ordinary output" (standard output, stdout, 1>, >) from "error output" (standard error, stderr, 2>)

```
prog > output 2> errors
```

- Null output:

```
> /dev/null
```

to get rid of some unwanted output  
e.g. error/warning messages from compilation/search  
(redirects it into a non-existent file)

- The null device exists on Windows too - link in notes

- To redirect script output to a file, from command line

```
script > file
```

- To redirect script output to a file from within the script, put this at start of script:

```
exec > file
```

- Then run:

```
script
```

- Program detect if output going to screen or not:

- Have you noticed this:

```
ls
```

(gives multi-column output)

```
ls > file
```

(gives single-column output)

```
ls | prog
```

(gives single-column output)

- You can detect if output is going to terminal, file or pipe, and adjust output accordingly.

- Shell script to detect where output is going:

```
if [ -t 1 ]
```

```
then
```

```
    echo stdout
```

```
else
```

```
    echo pipe or file
```

```
fi
```

- Test it:

```
prog
```

```
prog | cat
```

```
prog > file
```

- Advanced Bash-Scripting Guide - link in notes

- Redirection - link in notes

## Arguments and returns (more)

|         |                                                                                                                                                                                                                                |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \$1     | 1st argument                                                                                                                                                                                                                   |
| \$*     | all arguments                                                                                                                                                                                                                  |
|         |                                                                                                                                                                                                                                |
| \$0     | name of prog                                                                                                                                                                                                                   |
| \$#     | no. of args                                                                                                                                                                                                                    |
|         |                                                                                                                                                                                                                                |
| shift   | shift args leftwards<br>this is useful if you want to remove some of the first args, then<br>"shift" a couple of times,<br>and then do "for i in \$*" with the remaining args<br>e.g. grep (switches) (string) file1 ... filen |
|         |                                                                                                                                                                                                                                |
| exit 20 | exit with a return code                                                                                                                                                                                                        |
|         |                                                                                                                                                                                                                                |
| \$?     | return code of last prog executed<br>e.g. quiet grep:<br>grep > /dev/null<br>and then check \$?<br>though grep may have -q (quiet) option anyway                                                                               |

## Environment variables

- Like global vars for all programs.
- Note any environment vars that are declared within a program are local to that program only.

|                     |                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------|
| env                 |                                                                                                                |
| printenv            |                                                                                                                |
| set                 | may display shell functions too                                                                                |
|                     |                                                                                                                |
| var=value           | set environment variable<br>N.B. no spaces!                                                                    |
| echo var            | print the string "var"                                                                                         |
| echo \$var          | print value of environment variable                                                                            |
|                     |                                                                                                                |
| echo \$HOME         | get into the habit of using these<br>instead of the actual hard-coded values,<br>- makes scripts more portable |
|                     |                                                                                                                |
| echo path is \$PATH |                                                                                                                |
| echo \$USER         |                                                                                                                |

## uname and arch

|                 |                          |
|-----------------|--------------------------|
| echo `hostname` | recall <b>backquotes</b> |
|                 |                          |
| uname           | show hardware, OS, etc.  |
|                 |                          |
| arch            | same as "uname -m"       |
|                 |                          |
| echo `arch`     | recall <b>backquotes</b> |

- Uname - link in notes
- man uname - link in notes
- "arch" on DCU Linux may give x86\_64 or i686 - link in notes
- "arch" on DCU Solaris may give sun4 or i86pc - link in notes

- Example of using arch in config files:
  1. share the same set of files across a number of Unix/Linux systems running on different hardware.
  2. I collect binaries for each platform, but keep them in separate directories underneath the \$home/bin directory.
  3. Then at login I set the path to automatically include the correct directory.
    - e.g. On my C shell system I put this in the .cshrc file:
 

```
set path = ( $home/bin/`arch` ... )
```

## Strings and echo

|         |                                                 |
|---------|-------------------------------------------------|
| echo    | print something on screen, followed by new line |
| echo -n | print with no new line                          |

|             |                        |
|-------------|------------------------|
| printf      | print with no new line |
| printf "\n" | print with new line    |

On some platforms, echo -e exists (interpret special backslash chars)  
On DCU Linux:

|                        |                          |
|------------------------|--------------------------|
| echo -e "\n text \n\n" | print multiple new lines |
|------------------------|--------------------------|

```
echo "string"
echo 'string'
```

It is useful to have 2 choices for string - single quote and double quote.  
If using one for something else, surround with the other.  
e.g. To search for single quote in file:

|               |                                       |
|---------------|---------------------------------------|
| grep ' file   | syntax error (why?)                   |
| grep "'" file | surround with quotes and it works     |
| grep '"' file | to search for the double quote itself |

The 2 forms of string are not equal:

|                     |                           |
|---------------------|---------------------------|
| echo "--\$HOME--"   | --/users/group/humphrys-- |
| echo '--\$HOME--'   | --\$HOME--                |
| echo '--'\$HOME'--' | --/users/group/humphrys-- |



## File wildcards

```
echo *           echo all files
echo f*          all files beginning with f
echo */*         files in next layer
*/**/*           etc.
```

- Important to realise it is the shell that interprets "\*" and passes the result to echo or ls or your program. It is not actually echo or ls itself that parses it.

```
grep string *
```

```
# grep does not understand *
```

```
# but that's fine because grep does not actually RECEIVE *
```

```
# what happens is:
```

```
# the shell EXPANDS * to a list of files and passes these to grep
```

```
# so grep actually receives:
```

```
grep string f1 f2 .. fn
```

- To see that it is the shell that expands it, assign it to an environment variable. Try these:

```
echo *
echo "*"
```

```
x=*
echo $x
echo "$x"
```

```
x="*"
echo $x
echo "$x"
```

```
x=`echo *`
echo $x
echo "$x"
```

## CA170: Week 7

### Shell Utilities

#### grep

grep search for a string

grep string file

(output of prog) | grep string | grep otherstring

-i ignore case

-v return all lines that do NOT match

- grep
- man grep
- grep is Substring-oriented (will match inside a word).

Possible alternative:

grep -w

- grep is Line-oriented (will print whole line that matches).  
Not always good: Hit we want may spread across multiple lines, so grep won't find it. Or conversely, HTML file could be entire file on one line.  
Possible solution:
  - Pre-process the file.
  - If hit spread across multiple lines, use tr or sed to change all (or many) newlines to spaces.
  - If file all on one line, use same tools to introduce new lines at different points.
- Recursive grep:
  - This exists on some (but not all) Unix/Linux.  
grep -r string .  
grep -r --include="\*html" string .
  - Might be able to leave out the .
- More complex solution to long-line problem:
  - Use egrep to display from 0 to max 40 chars each side of the string:  
egrep -o ".{0,40}string.{0,40}" file
  - . means any char
  - 0 to max n of any char is .{0,n}

## string matching / regular expressions

|                 |               |
|-----------------|---------------|
| <code>^</code>  | start-of-line |
| <code>\$</code> | end-of-line   |
| <code>.</code>  | any character |

where "c" stands for the character:

|                         |                            |
|-------------------------|----------------------------|
| <code>c*</code>         | 0 or more instances of c   |
| <code>cc*</code>        | 1 or more instances of c   |
| <code>grep "  *"</code> | 1 or more spaces           |
| <code>.*</code>         | any sequence of characters |

where "c" has a special meaning, e.g. is \$ or ., etc:

|                        |                          |
|------------------------|--------------------------|
| <code>\c</code>        | the character itself     |
| <code>grep "\."</code> | the '.' character itself |

recall **the two forms of string**:

```
grep '\$' works (searches for the "$" char instead of end-of-line)
grep "$" fails (double quote treatment of $ is different to single quote
treatment of $)
grep "\\$" works
```

- Exercise: Go to [share/testsuite](#).
  - Find all lines in web pages containing "born".
  - Find all lines containing the "\$" symbol.
  - Find all lines containing "born" at start of line.
  - Find all lines containing start of line, then one space, then "born".
  - Find all lines containing start of line, then any number of spaces, then "born".
- Regular expressions - [link in notes](#)
  - POSIX syntax - [link in notes](#)
  - POSIX character classes - [link in notes](#)

## cut

**cut** extract columns or fields of text on command-line

To extract **columns** 30 to end of line of the ls listing:

```
ls -l | cut -c30-
```

In grep output, extract the 1st **field**, with delimiter ":"

```
grep string *html | cut -f1 -d':'
```

Extract the 2nd to end fields, with delimiter ":"

```
grep string *html | cut -f2- -d':'
```

- Why "-f2-" ?
- Why not "-f2" ?

## sed

sed "stream editor" - find and replace text on command-line (and other things)

sed 's|oldstring|newstring|' change first match on each line  
sed 's|oldstring|newstring|g' change all matches

'|' is just my choice of a separator.

Other people like '/'

We can actually use any character as a separator (whatever comes first after "s").

e.g. ls listing that highlights web pages:

```
ls -l | sed "s|\.html| [Web page]|"
```

e.g. ls listing that changes how my username appears:

```
ls -l | sed "s|$USER|ME|"
```

- Sed - link in notes
- man sed - link in notes
- sed FAQ - link in notes
  - How to do case-insensitive matching (can be tricky)
- sed examples - link in notes

## sed examples

- To insert a new line
  - How to insert new line - link in notes
  - On DCU Linux, put a new line in front of every string "www":

```
sed 's|www|\nwww|g'
```
  - (recognises "\n")
  - On some other platforms, need to do this:

```
sed 's|www|\nwww|g'
```

- To put new lines in front of and after every HTML tag
  - On DCVU Linux

```
sed 's|<|\n<|g' |
```

```
sed 's|>|\n>|g'
```

- Other platforms

```
sed 's|<|\n<|g' |
```

```
sed 's|>|\n>|g'
```

- To substitute back in the pattern we matched

```
# \( ... \) to mark a pattern
# \1 to reference it later
```

```
# e.g. change:
# (start of line)file.html: ...
# to:
# <a href=file.html>file.html</a>: ...
```

```
# search for:
# ^\(.*\.html\):
# change to:
# <a href=\1>\1</a>:
```

```
grep -i $1 *html |
```

```
sed -e "s|^\(.*\.html\):| <a href=\1>\1</a>: |g"
```

## tr

- tr - character substitutions

```
# change spaces to new lines:
```

```
cat file | tr ' ' '\n'
```

```
# convert Windows file format to Unix file format:
```

```
tr -d '\015'
```

- man tr
- uses character classes (numeric, alphanumeric, etc.)

## awk

- awk - a powerful pattern scanning and processing language

## dirname, basename

- useful cutting and pasting with filenames
- dirname - link in notes
- Basename - link in notes

```
$ echo $HOME
/users/group/me
```

```
$ dirname $HOME
/users/group
```

```
$ basename $HOME
me
```

```
$ dirname `dirname $HOME`
/users
```

## head and tail

- Head
  - Display the first 100 lines of the output:

```
grep string files | head -100
```
  - Note: When head gets the first 100 lines, the pipe closes and grep terminates.
  - As opposed to: Doing the entire grep and then taking the first 100.
  - To see this is true, run the program "yes" (which outputs an infinite number of lines) with head, and you will see it does stop:

```
yes | head -20
```
- Tail
  - Display the last 30 lines of the logfile:

```
cat logfile | tail -30
```

## date

```
date                                looks like: "Tue Feb 17 16:28:33 GMT 2009"
CURRENTDATE=`date`                remember backquotes
echo $CURRENTDATE
```

```
date "+%b %e"                      looks like: "Jan 21"
```

```
date "+%b.%e.log"                  can add things to the string
```

```
file=`date "+%b.%e.log"`
echo $file
```

- man date - link in notes
- Using date to get unique filename
  - Say web server in response to client needs to make a temporary file.
  - Use date to get a new filename that is unique to the current second:

```
timenow=`date +%H%M%S`
filename="/tmp/random.$timenow.txt"
```
  - Unique to second and nanosecond:

```
date "+%H.%M.%S.%N"
```
- Alternative ways of getting unique filename
  - Unique filename based on process ID:

```
filename=/tmp/random.$$ .txt
```

    - Would be different for each process.
  - Random number generator in Shell:

```
echo $RANDOM
```

    - This is a strange environment variable. It does not exist until you try to access it. Then it exists!

```
set | grep -i random
echo $RANDOM
set | grep -i random
echo $RANDOM
```

- About random-number generators:
  - A pseudo-random-number generator typically generates a series of random-looking numbers based on a start seed.
  - If the seed is identical, the algorithm will generate the same numbers.
  - So we want a different start seed each time we run.
  - Seed could be based on the current time (second and nanosecond).

|                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>XUKa"J&gt;FGh\$Jrmpt&amp;ka"GhgzJb}yPL_ 'NAojQrdZalzmqtni gYsypcd7^\$Zz&lt;P 5)L('d9oERKn@M\2~Tv.uyZt;P!rNizg Y9q~feZ.L2oE\_DXUVI}ziJ2EJ[fO(Vq &gt;X]{uHIijZ-. )m)6J{ }HPlP3&lt;R~ToamW Na5l9Cd:j@\ly!uqm\qUBCU26;3bD#&lt;q QqEYsn/]~s!Wu{))\$qCfi"#L0y0IEE3A  0li1Bb=D*I8LX]`oWizph&amp;!%@WKMD5&lt;</pre> | <input type="radio"/> Randomize the Seed Key<br><input checked="" type="radio"/> Freeze the Current Key<br><input type="text" value="10000"/> Range of Values<br><input type="text" value="10000"/> Count of Values<br><input type="button" value="Generate Random Numbers"/> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

The 256-character printable ASCII "SeedKey" region above can be copied and pasted for storage and reuse.

- JavaScript random number generation demo.
- Usage: Freeze the seed key. Set parameters. Then generate random numbers.
- Can view JavaScript code to see how it works.

## CA170: Week 8

### Search Engine Shell

#### How to write a search engine in 9 lines of Shell

- The following is a search engine for a website in 9 lines of Shell:

```
#!/bin/sh

echo "Content-type: text/html"
echo

echo '<html> <head> <title> Search results </title> </head> <body>'

argument=`echo "$QUERY_STRING" | sed "s|q=||"`

cd /users/homes/me/public_html

echo '<pre>'
grep -i "$argument" *html */*html | sed -e 's|<|\&lt;|g' | sed -e
's|>|\&gt;|g'
echo '</pre>'
```

#### Notes

1. This is an online program. It is a server-side CGI script.  
It accepts input through a HTML form.
2. "q=" assumes that your input variable is called "q" in the HTML form.
3. Your web directories need to be readable for the wildcard to work.
4. We pipe the result of grep into an ugly-looking sed command. This sed command is needed because there are HTML tags in the results returned by grep. These will be interpreted by your browser, displaying a mess.  
To just print the HTML tags without interpreting them, we need to pipe the results through a sed command that:
  - converts all < characters to &lt;;
  - converts all > characters to &gt;;
  - The command is tricky to write because "&" has special meaning to sed and must be escaped.

#### Some enhancements

- change the output so the user can actually click on the pages returned.
- Consider where there are spaces in the argument (multiple search words), etc.



### Some further enhancements

- If you have more than 2 levels of web pages you may write them out explicitly as `*/*/*/html` etc., or get a recursive grep, or use recursive find first to build the filespec:

```
cd /users/homes/me/public_html
```

```
filespec=`find . -type f -name "*html" | tr '\n' ' '`
```

```
grep -i "$argument" $filespec
```

- Since each search will be using the same file list, it would be more efficient to pre-build the list once, and cache it in a file, and then:

```
read filespec < filelist.txt
```

```
grep -i "$argument" $filespec
```

- The pages are not ranked in order of relevance, but only in the order in which `grep` finds them.
- Not easy to solve.

### My search engine started out like this

- My search engine started out as a few lines of Shell like the above (plus a C++ input pre-processor for Web input security).
- It has since been re-written in PHP, but there is still a `grep` at the core.
- Obviously a heavy-duty search engine would pre-index the files in advance, rather than `grep`-ing them on the spot. But a `grep` is perfectly fine for a site of less than, say, 5,000 pages.

See search engine lab

## CA170: Week 9

### What is an Operating System

#### Intro

- List of operating systems
- Comparison of operating systems

#### What is an Operating System?

1. Provide an environment in which programs and users can work.
  - Extract problems that are common to multiple programs and users on the system, and provide the solutions to them, so that programs and users do not have to.
    - e.g. The programmer writing a program does not need to know about the physical layout of the hard disk
    - he wants to be able to work with a concept called "files" (as does the user).
  - Not wanting programmers to have to reinvent the wheel each time:
    - File systems
      - Where does this file go on the physical disk? What if it grows bigger than the slot initially assigned to it?
    - User interfaces
      - If I move a window, how do I re-draw the window underneath? What pixels do I set to draw the letter 'A'?
  - These are not questions to be solved by each programmer.
  - They have other things to do. These are questions for the OS.
  - Network utilities - provide API for applications to use.
  - Bundling utilities so that users have many/most programs they need already installed. Editor, browser, calculator, image editor. Could go on.
2. Clever OS algorithms squeeze the most usage out of slow machines and limited resources
  - Wanting to run multiple programs on one machine, ideally overlapping in time. e.g. A web browser and a text editor. Take it for granted.
  - Wanting multiple users to be able to share one machine at the same time (web server, ssh to mainframe).
3. HIDE the reality of the system from the programs and users.
  - Hide the fact that my program is scattered all over memory, is being moved around in memory, and is being swapped out to disk.
  - Hide the fact that my file is physically scattered all over the disk, and is being moved around the disk.
  - Hide the fact that my program shares memory with lots of running system programs and multiple other user's programs, all of which are being moved around.
  - Hide the fact that my program is not actually running constantly, but is actually getting little timeslices of the CPU, in between it going off to service other programs.

### Principles that an OS will work by

1. Respond to the user interface as quick as possible. Everything else can wait.
2. Only load into memory the absolute minimum that we need to work with. We can load more if and when needed.
  - For example, just load the initial code to display a program and its menus. Don't load the code to handle specific menu items until they are called (which often never happens).
3. Only write to disk the absolute minimum we need to make the changes.
  - Example: Editing a 5 M file. Added a few lines to the end. Press "Save".
4. When deleting large areas of memory or disk, just put markers at the start of block and end of block showing it is free to be overwritten if and when needed in the future. There is no need to actually go through every location and scrub it clean.
  - Example: Delete a 10 G file.
  - See un-deleting files.

### "Never Enough"

- The whole history of Operating Systems / Computers can be summarised as:
- "Never enough".
- Never enough CPU speed.
- Never enough memory (RAM).
- Never enough disk space. (Perhaps solved for PCs. 10 T now less than \$300.)
- Never enough disk space on reliable, redundant, always-on, disk-always-spinning, high-use server. (Much more expensive. 10 T costs around \$10,000.)
- Never fast enough disk access.
- Never enough screen size.
- Never enough bandwidth.
- Never enough battery life time.

## PC Operating Systems

### PC Operating Systems (Single User)

- Microsoft dominance. Been that way for a long time.
  - Usage share of operating systems - link in notes
  - Desktop and laptop computers - link in notes
- 
- Desktop Operating System Market Share 2020 by netmarketshare.com.
- Do one month span only to get bar chart.
- Microsoft dominance, but not quite monopoly.
- 
- Desktop Operating System Market Share 2020.
- From StatCounter.

### PC Operating Systems

- DOS / Windows - link in notes
- Mac OS X - link in notes
- UNIX / Linux - link in notes
- Historical
  - OS/2 - link in notes
  - Mac OS - link in notes
- Comparison of operating systems - - link in notes
- Comparison of operating system kernels - link in notes
- Comparison of open-source operating systems - link in notes

## Server Operating Systems

### Mainframe / Server operating systems (multi-user)

- Split between UNIX/Linux and Microsoft. Been that way for a long time.
- Usage share of operating systems
  - Servers
  - Mainframes
- Usage of operating systems for websites 2020.
- Usage share of server operating systems for web servers only (not all servers/mainframes).
- "Unix" family includes Linux.
- IDC's Worldwide Quarterly Server Tracker tracks all servers (not just web servers)
  - Pay to view, so hard to link to.
  - In 2014 this had server OS share (by percentage of revenue):
    - Windows 45.7%
    - Linux 28.5%
    - Unix 13.6%
    - z/OS 8.0%
  - Anything more recent on a public link?

### Server operating systems

- mainframes / servers (multi-user),
- file servers (shared filespace in LAN),
- web servers
- UNIX / Linux
- Windows Server
- z/OS
- VM
- VMS

### Supercomputers

- Supercomputer operating systems
- TOP500 list
- Dominated by Linux.
- As at 2020, top 10 are all Linux.

## Mobile operating systems

### Mobile operating systems

- A formerly diverse market has settled into Android v. iOS.
- Market share listings may not agree because of differences in recording "OS of phones" v. "OS of phones that access websites"
- Usage share of operating systems
  - Mobile
- 
- Sales stats show a formerly diverse market ...
- 
- ... changing to Android dominance.
- 
- Mobile Operating System Market Share 2020 by netmarketshare.com.
- 
- Mobile Operating System Market Share 2020.
- From StatCounter.
- 

### Mobile operating systems

- Mobile operating systems (smartphone, PDA, tablet)
- Comparison of mobile operating systems
- iOS (Apple, closed source, UNIX family)
- Android (Google, open source, Linux, UNIX family)

### App stores

- App stores
- Comparison of mobile software distribution platforms
- App Store (iOS)
- Google Play (Android)
- Tablet Operating System Market Share 2020 by netmarketshare.com.

### Tablets

- Tablets
- Usage share of operating systems
- Tablets
- iOS and Android split the market
- i.e. Unix family dominates the market

## History Of Operating Systems

### History of Operating Systems

- History of Operating Systems
- Timeline of Operating Systems

#### (1) In 1940s-50s, Program = Computer

- At start, there were no Operating Systems.
- Computers were like the abstract model of a machine,
- running one program literally and nothing else.
- 1 user at a time
- 1 program at a time
- Programmer operates machine himself.
- Manually load program, run until crashed (dump memory and registers) or finished (print output), revise program, run again, or run next program.
- Interactive (great if you're the lone programmer) but CPU idle for long periods (e.g. while program being revised, or when program halts/crashes when programmer not watching).
- Long wait to use machine for other programmers.
- DRIVING FORCES FOR CHANGE:
- LOT MORE PROGRAMMERS WANTING TO USE MACHINE
- COMPUTERS EXPENSIVE (ANY CPU IDLE TIME BAD)

#### (2) In 1950s-60s, Operator hired to run computer.

- Programmers have to submit jobs, receive results later.
- Operator schedules jobs.
- Jobs still stored on sequential (tape) access (no random access medium yet).
- Sequential tapes of jobs are prepared by operator for the CPU to run through once.
- Resident monitor, always in memory (first OS).
- Doesn't decide order of jobs, but does co-ordinate their sequencing, automatic starting and terminating.
- Jobs come with control cards to say what they need to run.
- Downside for programmer: long queues, not interactive any more,
- if error in program may have to wait days to retry it.
- Have to think things through in advance!
- DRIVING FORCE FOR CHANGE:
- RANDOM (DISK) ACCESS BECOMES AVAILABLE.

#### (3) Pool of jobs on disk, random access.

- OS can now implement the human operator's algorithms in deciding sequence of jobs to run.
- Scheduling of jobs now totally automated (true OS).
- DRIVING FORCE FOR CHANGE:
- I/O DEVICE SPEED IS MUCH SLOWER THAN CPU SPEED,
- SO CPU STILL OFTEN IDLE WHILE I/O GOING ON.

#### (4) Some parallelisation of I/O and computation.

- Device controller is a piece of hardware that can work in parallel with the CPU.
- Not a parallel computer
- - it is just a specialised device for data transfer, not a general-purpose computer.

- Spooling - Print device controller is still copying data from disk to printer
- while CPU has already begun working on next job.
- Next job can begin while previous is still printing out.
- CPU can store output on disk if printer full and move on.
- DRIVING FORCES FOR CHANGE:
- WAIT TIMES STILL TOO LONG.
- Long jobs delay everything else. Run short jobs first?
- But if we have a small job that must be run once a day, then we cannot EVER run a program that will take 1.1 days.
- Also, program might do I/O half-way through its execution (rather than only at start/end).
- When program stops to wait for this I/O, CPU is idle.
- Maybe only short time, but it all adds up.

#### (5) Multi-programming.

- Multiple jobs in memory (and running) at the same time.
- Each memory space protected from each other.
- OS picks one, executes it for a while, stops (e.g. when program waits for input, or else randomly), picks other to run.
- CPU busy, but still batch model, not interactive.
- Scheduling:
- Job scheduling (or long-term scheduling)
- - which jobs get into memory at all
- CPU scheduling (or short-term scheduling)
- - which to run at each step (many may be ready)
- DRIVING FORCES FOR CHANGE:
- INCREASING NUMBER OF -USERS- WHO WANT TO
- INTERACT WITH PROGRAMS WHILE THEY ARE RUNNING.
- COMPUTERS CHEAPER - CHEAP DUMB TERMINALS AVAILABLE.
- HUMANS' TIME IS EXPENSIVE - DON'T WANT THEM TO WAIT

#### (6) 1970s-80s. Interactive time-sharing on mainframes

- Multi-programming where the program may be waiting on a USER.
- OS will in the meantime service another program, which may be interacting with another user.
- Result: Multiple users share the CPU.
- If the time-slicing is quick enough, they all feel as if they have their own dedicated machine!
- Programmers delighted - CPU kept busy, yet interactive again, just like in (1).
- User interaction at run-time allows a whole world of programs that were never possible before.
- In practice shared systems can get overloaded and slow down.
- DEC VT100 terminal (1978).
- This kind of terminal would be used to run programs on a mainframe shared with other users.
- Users can now interact with programs at run-time.
- DRIVING FORCES FOR CHANGE:
- REAL COMPUTERS GET VERY CHEAP

#### (7) 1980s. Standalone PCs.

- To some extent a return to simplicity of (5) and earlier
- - no traffic problems.
- 
- Internet still unimportant.



- LANs beginning to be used to coordinate file sharing.
- 
- Solitaire, bundled with Windows from 1990.
- - How office workers wasted time on PCs before the Internet.
- DRIVING FORCES FOR CHANGE:
- INTERNET BECOMES USABLE AND USEFUL.
- STANDALONE MACHINE NOW SEEN AS TOO ISOLATED.

#### (8) 1990s. Internet.

- Web is killer app for Internet 1993.
- Much use of information on shared remote web servers.
- Shared file systems.
- Shared email systems.
- Return to some of the problems of (6) - traffic problems.
- The shared mainframe returns:
- The Web.
- 
- DRIVING FORCES FOR CHANGE:
- BROADBAND (AT HOME).
- THEN LATER: MOBILE BROADBAND.

#### (9) 2000s

- Multimedia - video, audio, photo.
- YouTube (2005) probably the defining application of 2000-10 period.
- Mobile - Internet on mobile devices.
- Proper OSes (with proper file systems) on mobile devices.
- iPhone (2007) probably the defining invention of 2000-10 period
- 
- The decline of dialup, 2006-08 (in Ireland).
- 
- The mobile world before 2005:
- No or limited Internet.
- Image from here.
- \*\*video of steve jobs
- Steve Jobs introduces the iPhone at MacWorld 2007.
- The modern mobile world begins.
- Click through to video.
- Go to 5:20 where he introduces the modern concept of touch.

## CA170: Week 10

### OS and hardware

#### OS and hardware

- The fundamental job of an OS is to make the hardware usable for programs.
- To do this, it needs to understand the properties of its different forms of storage (in particular the access speeds and volumes).
- This is the "memory hierarchy" (see below).

#### Hardware v. Software

- First we consider what is a program anyway, and how can a program be implemented.

#### What is a Program?

- A list of instructions. A precise definition of what is to be done, as opposed to an English language description of what is to be done. A precise description is an algorithm, a recipe that can be followed blindly by a machine. A list of instructions to be executed one by one by a machine that has no memory of past instructions and no knowledge of future instructions.
- There are many processes that we observe in nature (animal vision, language use, consciousness) and invent in culture (calculating prime numbers, sorting a list). Some we have converted to algorithms. Some remain in the domain of an English language description.
- The Church-Turing thesis claims that "Any well-defined process can be written as an algorithm".

#### How is this Program, this list of instructions, to be encoded?

- What (if any) is the difference between the machine and the program?
- Many possibilities:
  1. Hardware (mechanical/electronic) is specially constructed to execute the algorithm. All you do is turn the machine on.  
e.g. Searle's "Chinese Room" thought experiment.  
(Hardware does not have to be general-purpose. Hardware can encode any algorithm at all.)
  2. Hardware is specially constructed to execute the algorithm, but there is some input data which is allowed to vary. This input data is read at run-time by the machine.  
e.g. Lots of machinery in factories and transport, calculators, clocks, (old) watches, (old) mobile phones.  
e.g. Many computer networking algorithms in routers (such as error-detection and error-correction) are done in hardware since they are run billions of times.  
Application-specific integrated circuit (ASIC)  
Bitcoin-specific ASICs are used to mine bitcoin.
  3. Hardware is a general-purpose device, and the program, as well as the input data, is read at run-time by the machine.  
e.g. Every normal computer.
- A general-purpose device implements a number of simple, low-level hardware instructions (the instruction set) that can combine (perhaps in the millions) to run any algorithm.
- Human programmer might write program in the hardware instructions themselves.
- Or might write program in High Level Language. Compiler translates this into low-level instructions for the particular hardware.

- 
- The complete Intel x86 instruction set (the start of the popular x86 CPU family).
- For full reference guide, see Intel manuals for the x86-64 architecture.
- If you write direct in the low-level instructions, program likely to be much more efficient. But you can only run the program on that particular hardware.
- If you write in a HLL, the same program can be recompiled into the low-level instruction set of a different machine, which is an automated process, and much easier than having to re-write the program from scratch.
- One HLL instruction like  $x := x+y+5$  may translate into many low-level instructions:
  - find the memory location represented by "x"
  - read from memory into a register
  - do the same with "y" into another register
  - carry out various arithmetic operations
  - retrieve the results from some further register
  - write back into a memory location

### CISC v RISC

- There is a body of theory, Computability theory, showing what set of low-level instructions you need to be able to run any algorithm.
- We can have pressure to:
  - Increase the number of hardware instructions - CISC model.  
Observe the system in operation. Anything done many times gets its own dedicated instruction in hardware. (This is the broad history of desktop Operating Systems).  
Example: Intel x86 family.
  - Decrease the number of hardware instructions - RISC model.  
Simpler CPU runs faster. May not matter if executes higher number of low-level instructions if runs much faster.  
Also lower power needs, less heat. Suitable for mobile, portable devices.  
Examples: ARM architecture.
- As Operating Systems have evolved over the years, they constantly redefine the boundary between what should be hardware and what should be software.

## The standard model for a Computer System

- We have different types of Computer memory.
- Permanent v. Temporary.
  1. Program is kept until needed on some permanent (or "non-volatile") medium (hard disk, flash drive, DVD, backup tape).

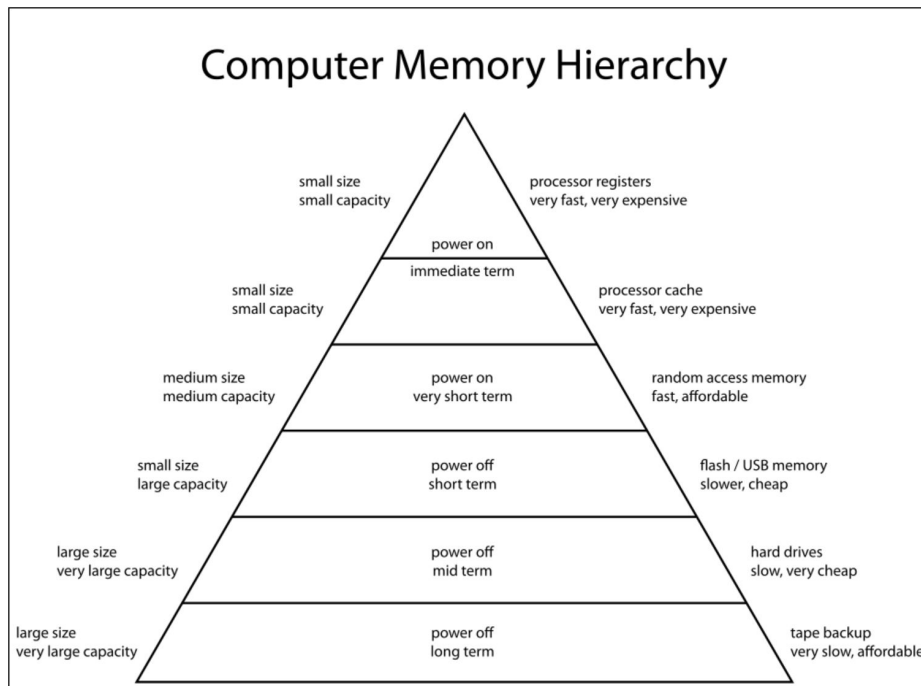
"Permanent" means retains data after power off.
  2. To run the program, it is loaded into some temporary (or "volatile") but faster medium (RAM).

Temporary data structures and variables are created, and worked with, in this temporary medium.
  3. In fact the CPU may not work directly with this medium, but require that for each instruction, data is read from RAM and loaded temporarily into an even faster, volatile medium (registers).

e.g. To implement  $X := X+1$ , where program variable X is stored at RAM location 100:

```
MOV  AX, [100]
INC  AX
MOV  [100], AX
```
  4. The results of the program are output either to some temporary or permanent medium.
  5. When the program terminates, its copy (the instructions) in the temporary medium is lost, along with all of its temporary data and variables. The long-term copy survives however, on the permanent medium it was originally read from.

## Memory hierarchy



- "Memory" = RAM.
  - An unstructured collection of locations, each of which is read-write, randomly-accessible.
  - = "working memory"
  - = the fast, temporary medium that we actually run programs in. Reset every time we restart the computer.
    - RAM may be (usually is) cached for speed. See Cache memory.
- "Disk" = Hard disk (moving parts), Solid state drive (flash memory, see below), USB flash drive, etc.
  - The permanent (non-volatile) medium.
  - A more structured space than memory, formatted with a file system in place, where different zones of the disk have names and are separated from each other.
  - The permanent medium that we store programs and data files in.
- "Read-only" memory needed for "bootstrapping" code:
  - There has always been a need for some "Read-only" (or at least hard-to-rewrite) non-volatile memory built-in to the computer to hold the basic "bootstrapping" code: (the basic code to load an operating system, find devices).
  - ROM - Read-only memory - used for this.
  - Some forms of ROM can only be written once.
  - EEPROM can be repeatedly re-written.
  - Flash memory (developed from EEPROM) used for bootstrapping now.
  - Flash memory can also be used as the main disk. You can format and structure a flash memory space, and keep a file system on it. See solid-state drives and USB flash drives.

### Up and down the memory hierarchy: Higher speed - Lower volume

- A typical computer: MacBook Pro.
- Spec (highest values):
- 4 M cache memory.
- 8 G RAM memory.
- 1 T disk.
- Shows the memory hierarchy of speed and volume.

### The multi-tier model is a practical necessity, not a mathematical necessity

- The multi-tier model is necessary to make machines work in practice, with the storage hardware that exists.
- Mathematically, a multi-tier system is not needed to run a program.
- Consider the following.
- If RAM was as fast as registers
  - If RAM was as fast as registers, the CPU could just operate on the RAM directly. No registers needed.  

`INC [100]`
  - Not very likely: This is not likely. A large, expandable memory space unlikely to ever be as fast as a small, dedicated memory space tightly integrated with the CPU.
  - Note that it is not just about speed: The circuitry does not exist to do arithmetic and logical operations on RAM. Only registers have this circuitry. RAM is currently designed only as a medium to send data to and receive data from registers. Adding this circuitry to every single RAM location would be a massive overhead, which is why the current, multi-tier architecture has evolved.
- If disk was as fast as RAM
  - If disk was as fast as RAM, then no need to load instructions into RAM from disk to be run. Run them direct from disk.
  - Not very likely: This is not likely. It seems likely though that a volatile medium will always be faster than a permanent one.
  - However, disk can be used as an overflow of RAM: Paging and Swapping.
- Thought experiment
  - The above is really just a thought experiment, to think about the possibilities for different architectures.
  - For the whole history of computers, we needed all the speed we could get, which is why these complex, multi-tiered machines have evolved.
  - Mathematically, this multi-tier system is not necessary. Only a single medium is needed.
  - From the engineering point of view, it is doubtful if a single-tier system will ever come into existence, for the reasons above. But it is useful to think about the possibilities for different architectures, especially when later we consider using disk as an overflow of memory, and memory as a cache for disk.
  - 2015 article on "The Machine", a Hewlett-Packard plan to get rid of the multi-tier system. "The Machine is designed to [scrap] the distinction between storage and memory. A single large store of memory based on HP's memristors will both hold data and make it available for the processor. Combining memory and storage isn't a new idea, but there hasn't yet been a nonvolatile memory technology fast enough to make it practical".

## Os Structure

## OS structure

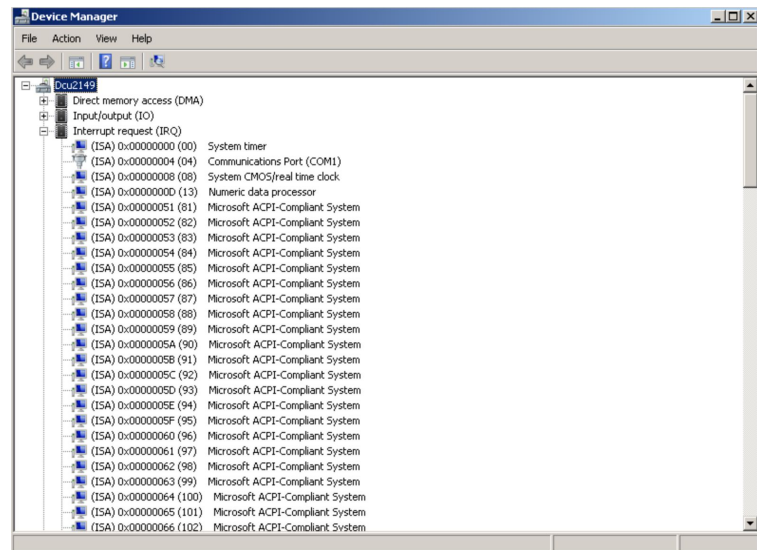
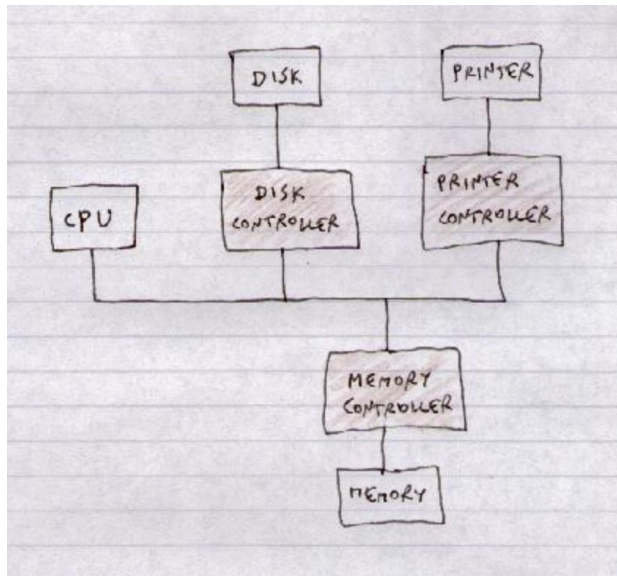
- "Process" = A program running. Could be multiple copies of the same program running. (Multiple processes.)

## Device speed

- See History of OS.
- What do we mean "I/O devices are slower than CPU" ?
- Easiest to see with user input. User types say 1 character per second average. The CPU takes 2 micro-sec to put character into a memory buffer. For 998 out of every 1000 micro-sec, the CPU is doing nothing.
- This holds even for non-user I/O. e.g. Read from disk is at very slow speed compared to speed at which CPU can work.
- Even read from RAM is at slow speed compared to speed at which CPU can work. CPU could be doing other jobs while the read is going on.

## Device controllers and Interrupts

- Device controller
  - Memory controller
  - Disk controller



- Device controllers operate in parallel.
- Can read/write devices and memory in parallel. Tell CPU when done.
- Instead of OS constantly polling device to see if done, device controller sends an interrupt.
- OS when it gets interrupt runs an interrupt handler.
- CPU is kept free (responsive to user, able to run other code) while device I/O is going on.
- Modern OS driven by interrupts.
- Interrupts are to do with the massive asymmetry between CPU speeds and device speeds. They are how the OS program runs on hardware with slow devices.

## Examples of Interrupts

- Completion of user input (every key press) causes interrupt.
- Sometimes each key press is handled just by hardware device controller, which builds up a buffer, and even handles line editing itself - and only when you (say) hit return at end of command-line is interrupt sent to OS.
- Imagine a typical command-line: Shell process may have waited hours for completion of that Input.
- Completion of any type of Input or Output.
- Deliberate system call in a program causes interrupt (essentially, the start of all I/O). System calls:
  - Process - exit, load, execute, wait, signal, alloc
  - Info - query date, set date, query-set system info
  - File - create, open, delete, read-write
  - Device - request, release, read-write
  - Communications - create connection, send-receive messages
- Also errors - Divide by zero causes interrupt.
- Bad memory access.
- Timer interrupts.

## Does an infinite loop cause an interrupt?

- Does an infinite loop quickly (or eventually) cause an interrupt?
- The Halting Problem (Turing, 1936).
- Might just be a long loop. We have no way of knowing.  
But still, even if long loop, control must switch occasionally - time-slicing.  
It is a timer interrupt that switches control. But loop runs forever, time-sli
- Unsolved problems in mathematics.
  - Note that if you could detect an infinite loop in general, then could solve all problems of the form:  

```
Does there exist a solution to f(n)
for n > t?
```
  - by asking the OS if the following:  

```
repeat
n := n+1
test solution
until solution
```
  - is an infinite loop, or just a long loop? Then our OS could solve many of the world's great mathematical problems.
  - Many mathematical problems can be phrased as infinite-loop problems.
- Note that many "infinite loops" actually terminate with a crash, because they are using up some resource each time round the loop. e.g. This program:

```
f ( int x )
{
f (x) ;
}

f (1) ;
```



- will eventually crash with a stack overflow.
- This program however:  

```
while true { }
```
- will run forever, time-sliced.

### Interrupts - Keeping the OS in charge

- Interrupt idea is a way of periodically keeping the OS in charge so nothing runs forever without reference to the OS. e.g. In time-slicing, periodic timer interrupt to give OS a chance to do something else with the CPU.
- Remember, the CPU is just looking for a stream of instructions to process, whether they come from the "OS" or from "user programs". When the OS "runs" a program, it points the CPU towards a stream of instructions in the user program and basically hands over control. How do we know program cannot now control CPU for ever?
- Note: Interrupt does not necessarily mean that OS immediately attends to the process that sent it (in the sense of giving it CPU time). It just means the OS is now aware that that process wants attention. The OS will note that its "state" has changed.

### CPU modes

- There must be a concept of an "OS-type program" (which can do anything, including scheduling ordinary programs) and an "ordinary program" (which is restricted in what it can do on the machine).
- The restrictions on the "ordinary program" are normally not a problem - they are basically just: "Other programs have to be able to run at the same time as you".
- This obviously makes sense in a multi-user system, but in fact it makes sense on a PC as well. When you're writing a program and you make an error, you don't want your crashed program to be able to crash the whole OS. You still want to be able to turn to the OS (which should still be responsive) and say "terminate that program".
- Code is executed either in kernel/system/monitor mode or in user mode.

```
boot in system mode, load OS
when run program, switch to user mode
```

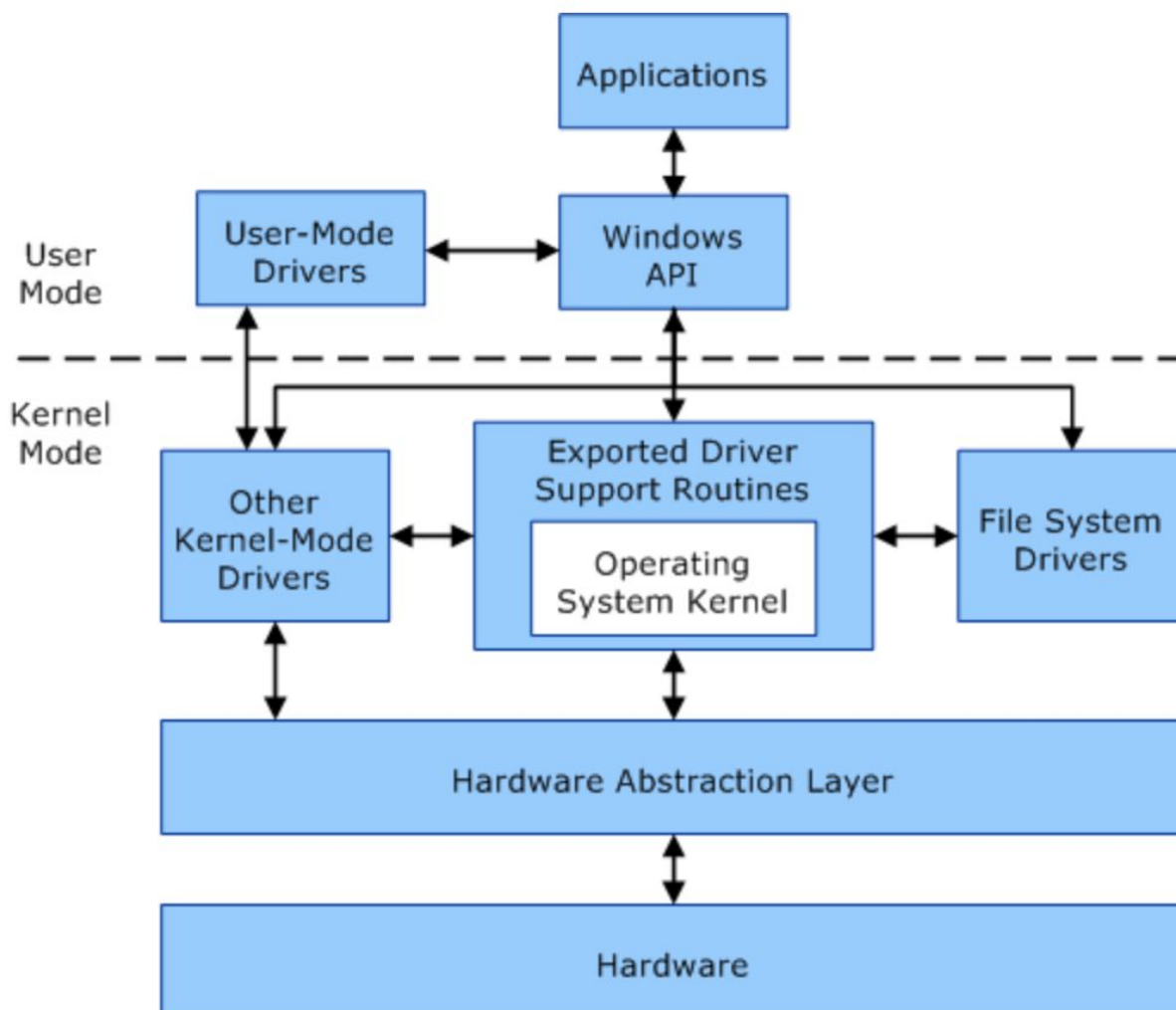
```
when interrupt, switch to system mode
and jump to OS code
```

```
when resume, switch back to user mode
and return to next instruction in user code
```

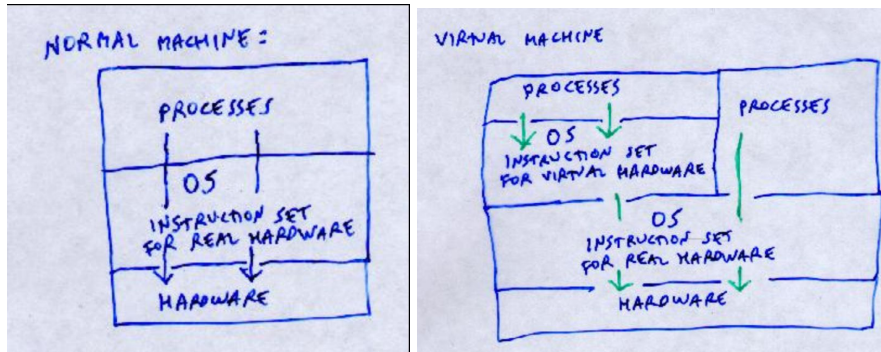
- Privileged instructions can only be executed in system mode.
- e.g. Perhaps any I/O at all - user can't do I/O, user has to ask OS to do I/O for it (OS will co-ordinate it with the I/O of other processes).
- On Unix/Linux, see these commands:
  - su
  - sudo

## The kernel is the part of OS that is not scheduled

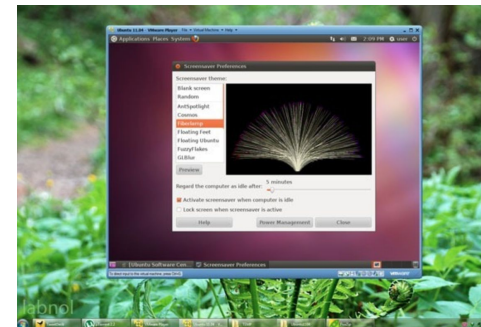
- Consider the different types of code, OS or applications:
  - Parts of OS that are not scheduled themselves. Are always in memory. Operate in kernel/system mode.  
This part of the OS is called the kernel.  
Includes:
    - memory management
    - process scheduling
    - device I/O
  - Parts of OS that can be scheduled. Come in and out of memory. Operate in user mode.  
Includes:
    - Command-line
    - GUI
    - OS utilitiesParts of OS that could be in either of the above:
    - file system
    - network interface
  - Applications. All scheduled. Come in and out of memory. Operate in user mode.



## Virtual Machine (VM)



- Virtual machine
- IBM VM "Virtual Machine" OS, since 1972.
- VM idea built-in. Can run a multi-user OS in the user space of a single user of VM.
- Emulator - Programs from Operating System 1 or CPU 1 running on Operating System 2 or CPU 2.
- VMware - Run Linux inside Windows. Run another Windows inside Windows. etc.
- I use this kind of thing in DCU labs to teach system administration of an imaginary multi-machine network all running on one machine.



## Java Virtual Machine

- Java is a language designed for a machine that does not exist, but that can be emulated on top of almost any machine. Java runs on a virtual piece of "hardware", the Java virtual machine.
- Promises:
  1. Portability. Write an application once, run everywhere.
  2. Run code from Internet on client-side. Even though from a site with different hardware, they will run on your hardware.
- Java is a HLL. It is "compiled" to a "virtual" instruction set called Java bytecodes. These run on Java VM, where they are actually mapped to native hardware instructions.
  - Java applets (client-side) - Written in Java, compiled bytecodes transmitted.

## What language is OS written in?

- OS'es are written mainly in HLL, with some Assembly in Kernel.
- Assembly fast but hardware-specific.
- HLL much more portable, much easier to write/debug/change/maintain.
- UNIX / Linux
  - Mainly written in C.
  - Some parts in Assembly.
- Windows
  - Mainly written in C, C++.
  - Some parts in Assembly.
- C / C++ allows you directly embed Assembly instructions within it.
- Comparison of open-source operating systems shows kernel programming language.
- Comparison of mobile operating systems shows languages programmed in.

## Is Assembly needed?

- Assembly is faster. But not needed everywhere.
- Perhaps only 5 % of the code is performance-critical. Things executed constantly - Interrupt handler, Short-term CPU scheduler.
- Other 95 % can be HLL for portability. 5 % can be machine-specific - needs a rewrite for each new hardware.
- Best to find out what is important to performance rather than pre-judge. (In which loops does the system really spend its time? You might be surprised.)
- Performance profiling
- 
- Another reason to stick with HLL:
- Modern compilers have smart optimisers:
- Compiler often has a -optimise switch to analyse the HLL code to generate faster Assembly code.

```
int __init_or_module do_one_initcall(initcall_t fn)
{
    int count = preempt_count();
    int ret;
    char msgbuf[64];

    if (initcall_blacklisted(fn))
        return -EPERM;

    if (initcall_debug)
        ret = do_one_initcall_debug(fn);
    else
        ret = fn();

    msgbuf[0] = 0;

    if (preempt_count() != count) {
        sprintf(msgbuf, "preemption imbalance ");
        preempt_count_set(count);
    }
    if (irqs_disabled()) {
        strlcat(msgbuf, "disabled interrupts ", sizeof(msgbuf));
        local_irq_enable();
    }
    WARN(msgbuf[0], "initcall %pF returned with %s\n", fn, msgbuf);

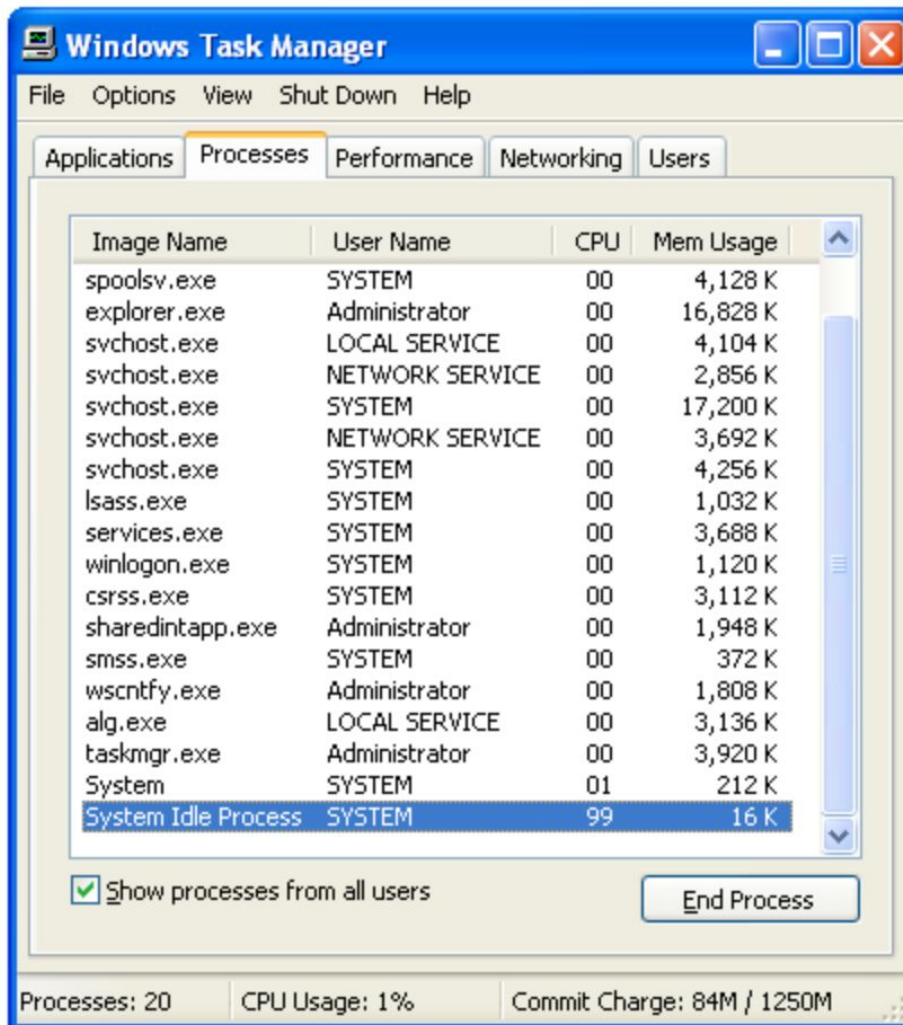
    return ret;
}
```

## Processes

### Processes

- Programs sit on disk. Maybe not run for years.
- Process - An instance of the program running.
- 1 user with 2 instances of text editor running - 1 program, 2 processes, each with their own memory space.
- 100 users with 100 instances of web browser running, all running the same copy from `/usr/bin/firefox` - 1 program, 100 processes, each with their own memory space.
- Processes (`ps`) in Linux

### Windows processes (Task Manager)

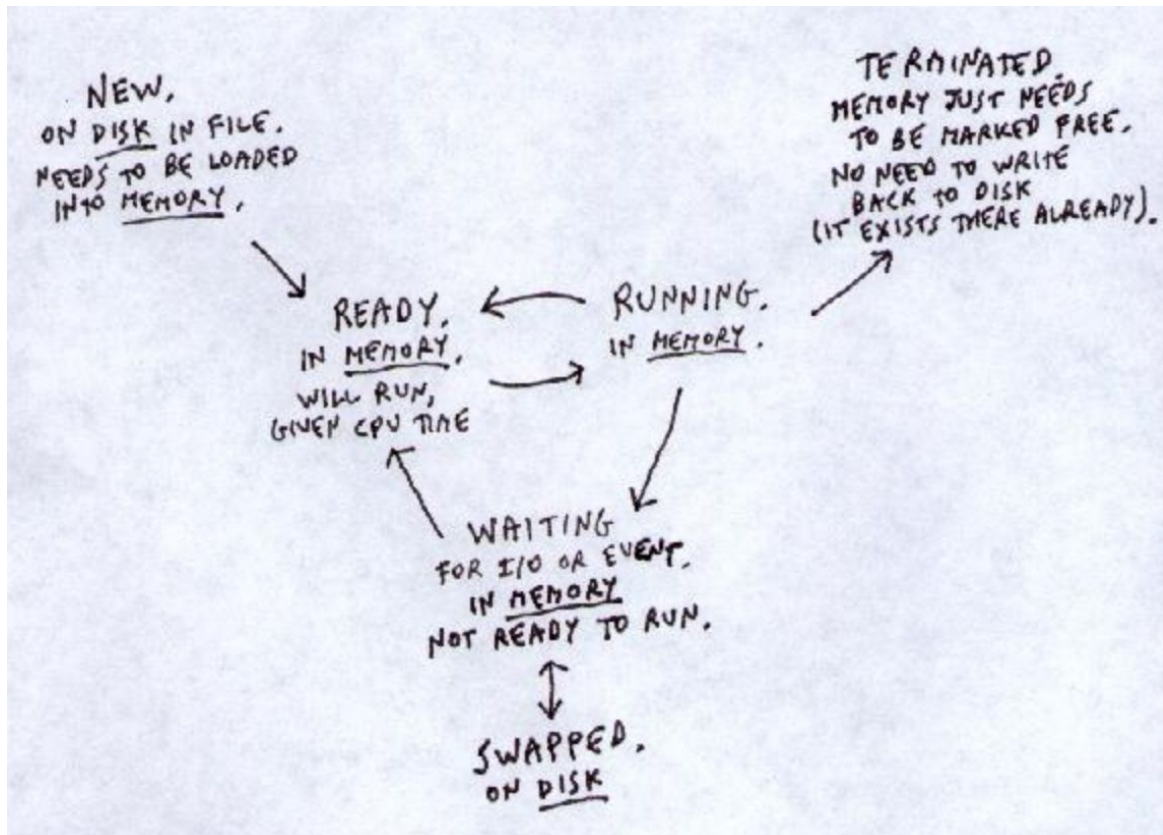


- 
- List processes in Task Manager.
- This is in fact Windows XP. From here.
- When the system is running slow, look for software updates running.
  - Some examples:
  - `trustedinstaller.exe` - Windows update
  - `msiexec.exe` - Windows update
  - `ARMsvc.exe` - Adobe update
- Sysinternals page at Microsoft has many tools to download, including:
  - Process Explorer - link in notes



## Process State

- Process state
- Loaded from disk (permanent, power-off storage) into memory (RAM).
- Recall Memory hierarchy
- Starts running.
- May pause when doing disk I/O, waiting on user event, etc. Does not always have work for CPU to do.
- May pause for long time.



- 
- READY to RUNNING - Short-term (CPU) scheduler decides if you can run now.
- RUNNING to WAITING - I/O or event wait.
- WAITING to READY - I/O or event completion.

## CPU scheduling (Time-slicing)

- RUNNING to READY - Why stop a process that is happily running?
- To allow the OS do something else with the CPU. e.g. To allow the OS give CPU time to other process for a while.
- This time-slicing by the Short-term (CPU) scheduler is how we run many processes at the same time.
- Most processes look like:
  - Short period of calculation (short "CPU burst")
  - Long period of I/O
  - Short period of calculation
  - Long period of I/O
  - ..
- i.e. Most processes regularly "vote themselves" off the Running list anyway. - We don't have to force them to go.

## Swapping

- It is often the case that processes are idle for a long time. e.g. A GUI process waiting on user input, that the user has not looked at in hours.
- If running short on RAM, and process has not run for a long time, it may be removed from RAM to disk. (Or parts of it may.)
- This is called Swapping to disk.
- It is the running copy on disk. Not the same thing as the original program file.
- If you click on the program again, there may be a delay as the OS gets it back from disk.
- Swapping is integrated with Paging

## Threads

- Threads
- 2 (or more) flows of control in the same process. 2 (or more) instruction counters. Same memory (same data and code space).
- e.g. Web server - Each request is a separate thread, all running in parallel until request completed.
- e.g. Web browser - multiple windows downloading at same time. Each is a separate thread in same process.
- On Linux, ps can list threads.

```
see all processes and threads:  
ps -ALf
```

```
LWP = thread id  
note multiple threads within same process id PID  
NLWP = no. of threads in this process
```

```
sort by processes with the most threads:  
ps -ALf | sort -k6 -n
```

```
show threads for one process:  
ps -Lf -p (PID)
```

- Scheduling of threads: OS may do the CPU scheduling between threads (kernel-level threads). Or program designer may implement their own (user-level threads).
- Up to program designer to make sure threads don't interfere with each other (or at least, only interfere in the ways he intends). Maybe nice to have OS protect us against incompetent thread design, but how can it? Note OS does not (can not) protect against lots of types of incompetence, e.g. infinite loop.
- Protection:  
Threads - 1 user - friendly (though might be incompetent).  
Processes - multiple users - hostile.

## Multiple cores

- Modern computers have multiple cores (multiple processing units, executing instructions in parallel).
- Modern OS will schedule threads and processes across multiple cores.
- Processor affinity - The idea that once a process or thread starts running on one core, it is often more efficient to leave it there (and its data in the core's cache memory) rather than change cores.
- On Linux we can look at multiple core use from command-line using ps and top and other commands
- See next page

## Event logging

- Event logging: The OS logs (in a file) events connected to processes.
  - Event logging
  - Windows Event Viewer.
- Unix/Linux logs
  - Unix syslog.
  - On Unix/Linux, various text and binary format logfiles are in:  
`/var/log`
  - fail2ban logfile is plain text  
`tail -20 fail2ban.log`
- Log of user logins on Unix/Linux
  - Record of user logins, stored in binary file:  
`/var/log/lastlog`
  - Can see them with lastlog command. (Same name as file!)
  - All users who logged in in last 2 days:  
`lastlog -t 2`  
`lastlog -t 2 | sort -k4`



## Multiple cores

Display number of cores available:

=====

```
$ nproc
```

```
4
```

Run process using core n (where n is 0 to 3 here):

=====

```
$ taskset -c n prog
```

See what processes are currently using what cores:

=====

```
$ ps -Ao user,pid,ppid,comm,psr
```

| USER     | PID   | PPID  | COMMAND     | PSR |
|----------|-------|-------|-------------|-----|
| root     | 1     | 0     | init        | 0   |
| root     | 2     | 0     | kthreadd    | 2   |
| root     | 3     | 2     | ksoftirqd/0 | 0   |
| root     | 25    | 2     | watchdog/3  | 3   |
| root     | 328   | 1     | udevd       | 3   |
| humphrys | 29858 | 29855 | sshd        | 0   |
| humphrys | 29859 | 29858 | cs          | 0   |
| humphrys | 32373 | 29859 | ps          | 1   |
| humphrys | 32374 | 29859 | grep        | 1   |

# PSR will be 0 to 3 here

Query if process is bound to a core:

=====

```
$ taskset -p 3
```

```
pid 3's current affinity mask: 1
```

```
$ taskset -p 25
```

```
pid 25's current affinity mask: 8
```

# 0001 (1) - bound to core 0 (see process 3 command /0)

# 0010 (2) - bound to core 1

# 0100 (4) - bound to core 2

# 1000 (8) - bound to core 3 (see process 25 command /3)

```
$ taskset -p 1
```

```
pid 1's current affinity mask: f
```

# 1111 (f) - can run on all 4 cores, not tied to any core

See load on different cores:

=====

\$ top

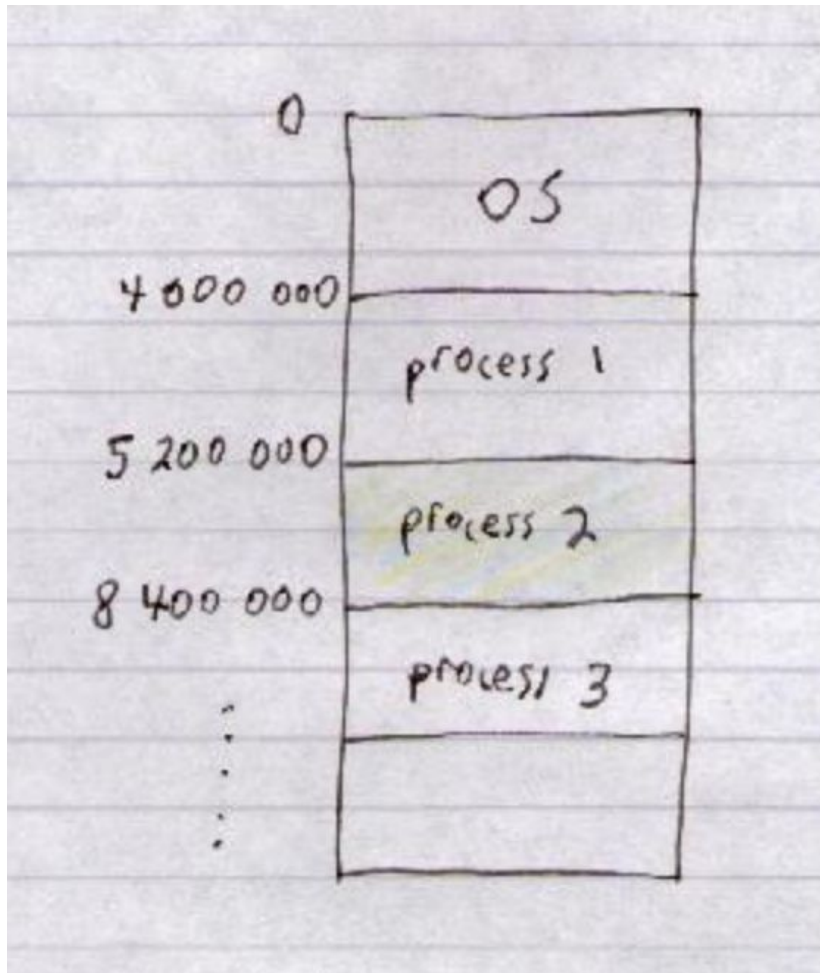
(and then press 1)

## CA170: Week 11

### Memory

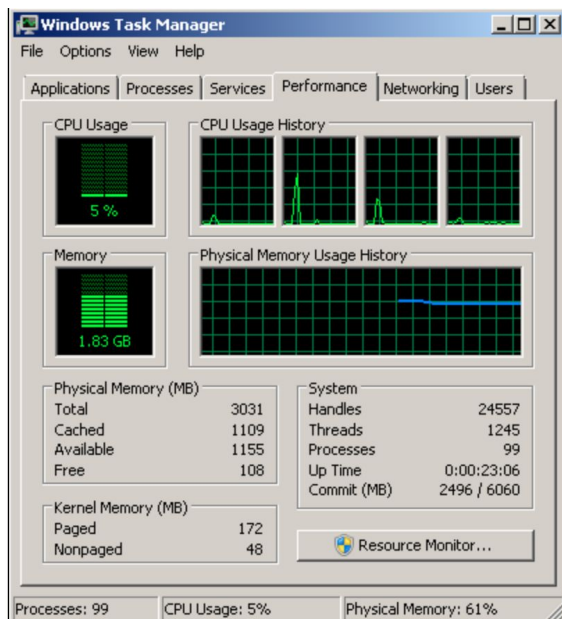
#### Memory Protection

- Basic idea is that there is such a thing as a "bad" memory access.
- User process runs in a different memory space to OS process (and other processes).
- On multi-process system (whether single-user or multi-user) each process has its own memory space in which (read-only) code and (read-write) data live.
  - i.e. Each process has defined limits to its memory space:

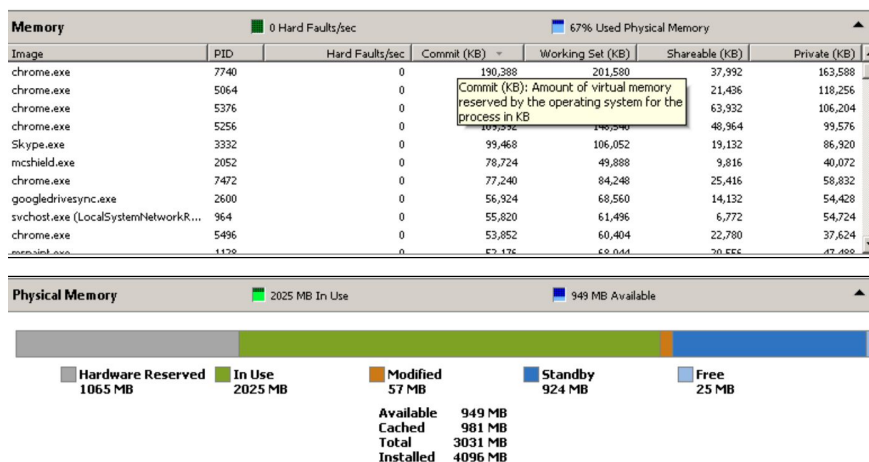


- Q. Even if there are no malicious people, process memory areas need to be protected from each other - Why?
- Above, each process has two registers - a base register and a limit register.
  - e.g. The base register for process 2 has contents 5200000. Its limit register has contents 3200000.
  - And then every memory reference is checked against these limits.
  - A simple model would look like this:  
When reference is made to memory location x:  
  
if x resolves to between base and (base+limit)  
    return pointer to memory location in RAM  
else  
    OS error interrupt
- Q. Load values into base or limit registers are privileged instructions. Why?
  - As we shall see, in fact memory protection has evolved far beyond this simple model.

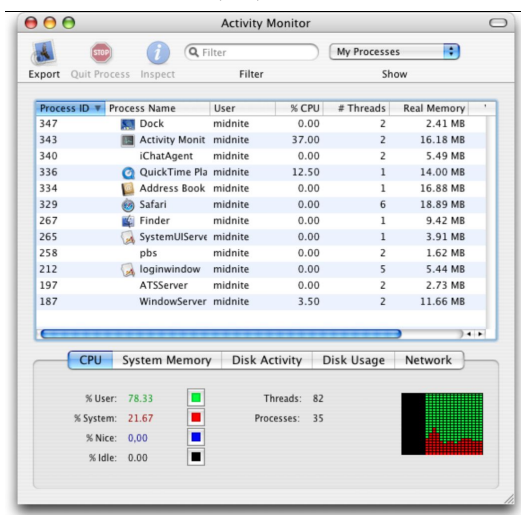
## Map of memory (Task Manager)



- Looking at memory use on Windows in Task Manager.



- Click "Resource Monitor" in the above and you get further breakdowns. (See "Memory" tab)
- Hover over items to see definitions.
- "Standby" here (in Windows 7) is really free RAM. See explanation later.
- See Processes (ps) in Linux/Unix



- Apple OS X has Activity Monitor.
- OS X is Unix family.
- You can also use ps on the Apple Terminal (command-line).
- Image from here. Creative Commons.

## Compile-time, Load-time and Run-time

- The transformations made to the program from the code written by a human to instructions executed by the CPU.
- Compiled program (e.g. typical use of C++, Java):
  - Program typically written "in factory" in HLL, with comments, English-like syntax and variable names, macros and other aids to the programmer.
  - Program is then compiled "in factory" into an efficient, machine-readable version, with all of the above stripped, optimisations made, and everything at "compile-time" translated and resolved as much as possible, so as little as possible needs to be done when it is run.
  - At different times, on different machines, and with different other programs already running, the user will "launch" or "run" a copy of this compiled program. Any further translation that could not be done at compile-time will now be done at this "load-time". Then the algorithm itself actually starts to run.
  - Any change that has to be done while the algorithm has already started to run, is a change at "run-time".
- Interpreted program (e.g. typical use of Javascript, Shell):
  - No compilation step. Source code itself is read at "load-time".

## Memory mapping (or binding)

- Consider what happens with the memory allocation for a program's global variables.
- Programmer defines global variable "x". Refers to "x" throughout his High-Level Language code:

```
do 10 times
    print(x)
    x := x+7
```
- Clearly when the program is running, some memory location will be set aside for x and the actual machine instructions will be:

```
do 10 times
    read contents of memory location 7604 and print them
    read contents of loc 7604, add 7, store result back in loc 7604
```
- How "x" maps to "7604" can happen in many ways:
  1. Source code: The programmer maps x to 7604 in the source code.
  2. Compile-time: If x is mapped to 7604 at this point (or before) then the program can only run in a certain memory location. (e.g. DOS - single user, single process)
  3. Load-time: The compiler might map x to "start of program + 604". Then when the program is launched, the OS examines memory, decides to run the program in a space starting at location 7000, resolves all the addresses to absolute numerical addresses, and starts running.
  4. Run-time: Even after the program has started, we may change the mapping (move the program, or even just bits of it, in memory), so that next time round the loop it is:

```
read contents of memory location 510 and print them
read contents of loc 510, add 7, store result back in loc
510
```

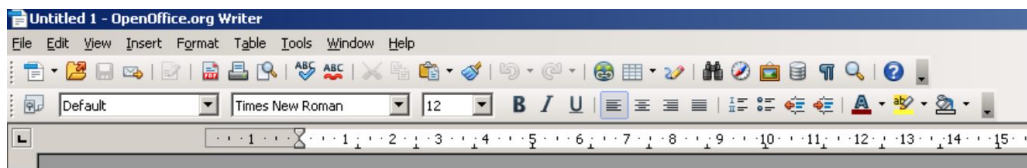
Obviously, OS has to manage this!  
User can't. Programmer can't. Even Compiler designer can't.

## Questions

- Question - Why is it not practical for the OS to say to a user on a single-user system: "I'm sorry, another program is occupying memory space 7000-8000. Please terminate the other program before running your program"
- Question - Why is it not practical for the OS to say to a user on a multi-user system: "I'm sorry, another program is occupying memory space 7000-8000. Please wait until the other program terminates before running your program"
- Question - Why can't program writers simply agree on how to divide up the space? Some programs will take memory locations 7000-8000. Other programs will take locations 8000-9000, etc.
- Question - The user's program has started. The OS needs to change what memory locations things map to, but binding is done at load-time. Why is it not practical for the OS to say to the user: "Please reload your program so I can re-do the binding"
- Question - Why is it not practical to say to a user: "Please recompile your program"

## Partial load of programs (desired)

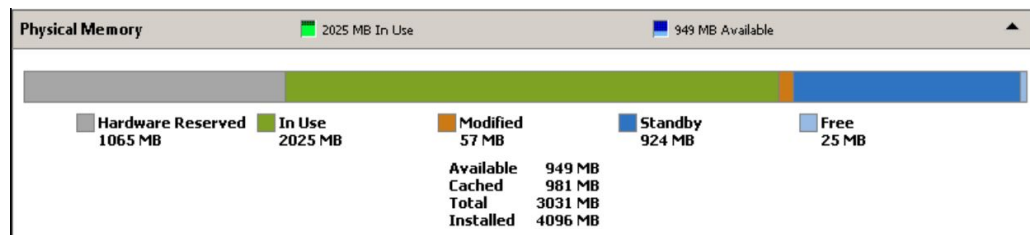
- Consider: Does the whole program have to be loaded into memory when it is run?
- Many applications contain vast amounts of functionality that is rarely or never used.
- e.g. Run web browser for years. Never once click on some menu options.
- But was the code behind those menu options loaded into memory every single time, for years?
- How can we load only part of a program?



- Many applications contain functionality that is rarely or never used.

## Pre-load programs and keep old ones in memory (desired)

- In apparent contrast to only loading part of a program into RAM before starting execution is the following idea, where we pre-load things (and keep old things) in memory that we may not even need



- Above we saw this typical memory map of a Windows 7 system.
- We saw there is not much free memory, but I said that "Standby" is really free memory.
- I will now explain this.
  - In the "Standby" section, the OS has filled RAM with stuff we may not even need:
    1. It keeps old code and data that may be needed again.
    2. It may pre-load commonly used programs into memory in case they are run.
      - See Windows Prefetch and SuperFetch algorithms.
      - Info on recently-run programs stored in .PF files in something like C:\Windows\Prefetch
- This is a good idea. First, this RAM is actually still free for use if needed. It is not being reserved. When more RAM needed, it can be taken from here.
- Second, pre-loads can be done when the system is not busy and need not detract from the user experience. Keeping old data in RAM of course costs nothing.
- Third, if the old data is needed again, or if those common programs are run, they are already in RAM and will run straight away without reading from disk. If they are not needed, there is no problem.
- Why fill RAM with stuff we may not need? For speed. This is the "unused RAM is wasted RAM" idea.

## Contiguous memory allocation

- Memory management
- Processes are large. Take up a lot of memory.
- Imagine if this has to be contiguous - all in one unbroken chunk from memory location A to location A+n.
- End up with a memory map like this:

| Start address | End address | Process    |
|---------------|-------------|------------|
| 0             | 1000        | OS         |
| 1000          | 1200        | free       |
| 1200          | 2000        | Process 11 |
| 2000          | 2100        | Process 12 |
| 2100          | 2400        | Process 13 |
| 2400          | 3300        | free       |
| 3300          | 3800        | Process 3  |
| 3800          | 4500        | free       |

- Problems with contiguous memory spaces:
  1. Programs starting and ending all the time. End up all over memory. Will have gaps everywhere.
  2. When load program, have to find contiguous space big enough to hold it.
  3. "Holes" develop in memory. Lots of memory free but not contiguous so can't be used.
  4. When process asks for more memory at run-time, might not be any free above it. (See Process above.)
  5. Difficult to load only part of the program at startup as opposed to whole thing (must reserve big amount of space above/below it).
  6. Difficult to swap out only part of a process as opposed to whole thing, unless we have some mechanism for identifying parts of a process.

## Paging

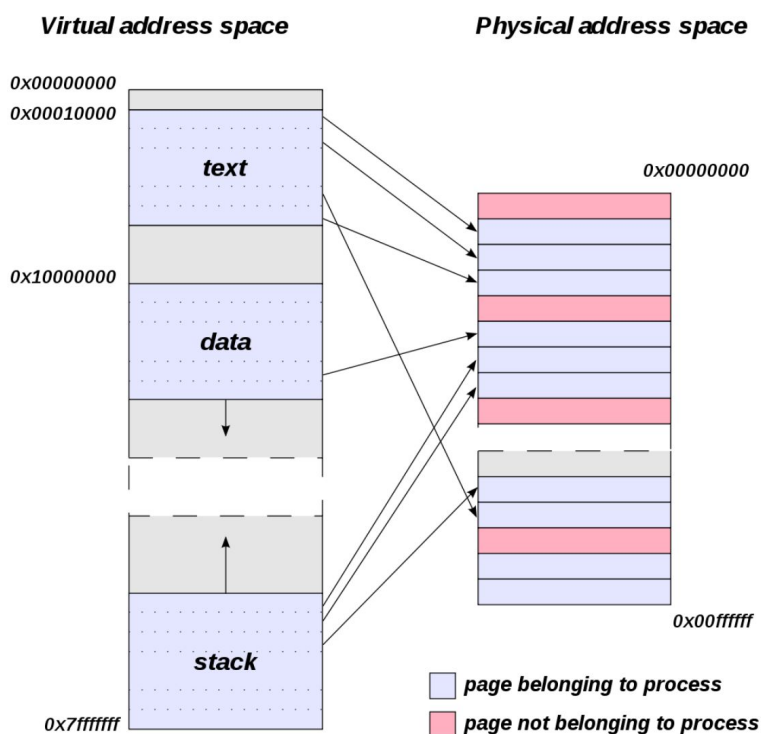
- The modern OS does Paging.
- Non-contiguous memory spaces.
- Program broken (automatically by OS) into lots of small chunks that flow like water into every crack and gap in memory, and out to disk (swapping) and back in.



## Memory mapping with paging

- Program logical (or virtual) memory space is divided into pages.
- Pages are of fixed size, say 4 K.
- On Linux can query page size at command line with getconf:  

```
$ getconf PAGESIZE  
4096
```
- Some architectures support multiple page sizes (pages, large pages, huge pages).
- A program references a memory location in its virtual address space as an offset within a particular page.
- Physical memory is divided into frames, of same size as pages, so that pages can be loaded into the frames.
- Program loaded into memory. Each page is loaded into a frame. Crucially, the frame can be anywhere in memory. Any free frame will do.
- Page table shows where the program is.



- Paging means the program can be physically located all over memory.
- From here.
- Example
- e.g. Program has logical memory space: page 0, ..., page 3. Physical memory frame 0, ..., frame 7. Page table:

```
page -> frame  
0 -> 1  
1 -> 4  
2 -> 3  
3 -> 7
```

- Don't have to be contiguous, don't even have to be in order.  
Page 0, offset 2, maps to: Frame 1, offset 2,  
Page 3, offset n, maps to: Frame 7, offset n,  
etc.

## Paging advantages

- Holes and fragmented memory are no problem.
- When load program, do not need to search for contiguous space, only enough memory in different locations.
- If a bit of memory is available anywhere, program can use it.
- Can easily ask for more memory, in any location.
- Can easily load only parts of program.
- Can easily swap only parts of process to disk.
- Makes maximal use of memory. Indeed, we would find modern machines almost unusable without paging.
- Separation of process memory from other processes. Process cannot address memory it doesn't own. Has no language to even describe such memory.
- Paging disadvantages:
  - Overhead - have to resolve addresses at run-time.

## Paging and Swapping (Page faults)

- Swapping pages out to disk:
- Program addresses a logical space. This logical space maps to pages scattered all over memory and on disk. Much of program can be left on disk if not needed. OS hides this from user and programmer.
- Only need: Active processes in memory.
- In fact, only need: Active bits of active processes in memory.
- OS decides what bits of programs stay in memory at any given time.
- If page not in memory it is not an error, OS just fetches it. This is called a Page fault.
- Not really: Disk as extra memory.
  - Disk I/O is slow. So using disk just as "extra memory" is not really the idea.
  - Yes it is extra memory, but program runs much slower if too much disk I/O.
  - Recall Memory hierarchy.
- More like: Disk as overflow of memory.
  - What we really use disk for is as an overflow/buffer/cache of memory. Use it so we don't have to worry about the exact size of memory - worry that launching the next program will cause OS to crash. Instead the system slows down gracefully as we run more progs. (More page faults, more disk I/O, system slows, user closes some programs.)
  - Without using disk as overflow, when we run out of memory there will be sudden and catastrophic crash of the system

## Major and minor page faults

- Minor (soft) page fault - Page is actually in RAM though was not marked as part of the "working set" for this process.
- e.g. Page was pre-fetched in case needed.
- e.g. Page was brought into RAM for another process.
- e.g. Old page marked free but not overwritten.
- Can resolve minor page fault without disk read.
- Major (hard) page fault - Need a disk read. Takes much longer to return.

## Viewing page faults

- On Linux we can look at major and minor page faults using ps:

```
# show min_flt
# and maj_flt
```

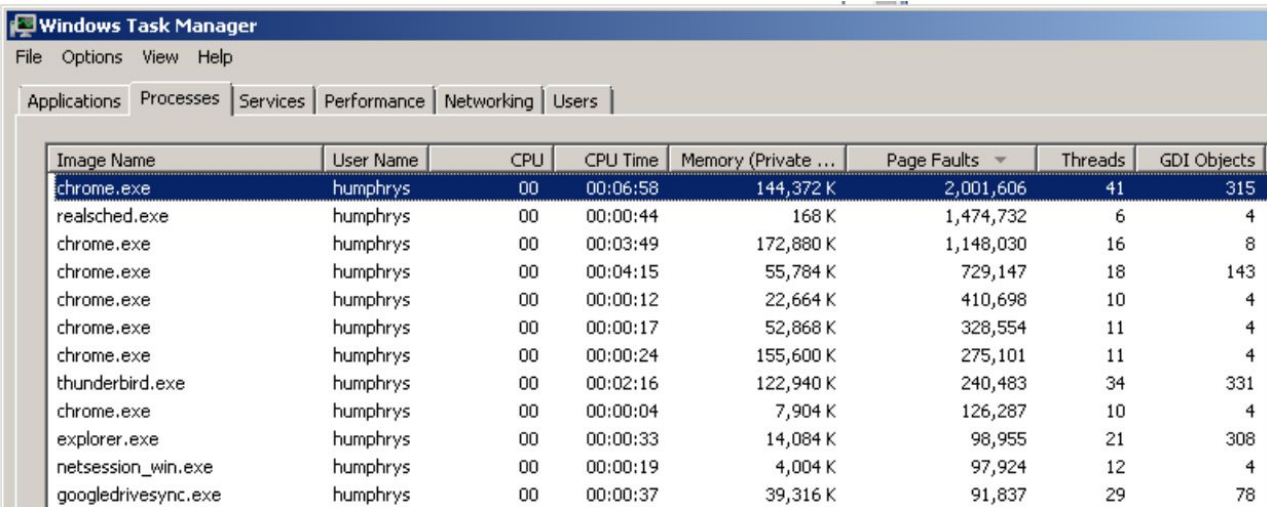
```
$ ps -A -o user,pid,ppid,min_flt,maj_flt,comm,args
```

```
# sorted descending by number of minor page faults:
```

```
$ ps -A -o user,pid,ppid,min_flt,maj_flt,comm,args | sort -nr -k 4
```

| USER | PID   | PPID | MINFL    | MAJFL | COMMAND                  | COMMAND          |
|------|-------|------|----------|-------|--------------------------|------------------|
| root | 1198  | 1    | 20509703 | 0     | xe-daemon                | /bin/bash        |
|      |       |      |          |       | /usr/sbin/xe-daemon      |                  |
| root | 2920  | 1    | 10267099 | 0     | sshd                     | /usr/sbin/sshd   |
| root | 3172  | 1    | 6000382  | 1     | xinetd                   | /usr/sbin/xinetd |
| root | 11034 | 1    | 1176904  | 0     | fail2ban-server          | /usr/bin/python  |
|      |       |      |          |       | /usr/bin/fail2ban-server |                  |
| root | 5293  | 1    | 975768   | 28    | nscd                     | /usr/sbin/nscd   |
| root | 3149  | 1    | 193727   | 0     | cron                     | /usr/sbin/cron   |

- On Windows we can look at combined page faults using Windows Task Manager:

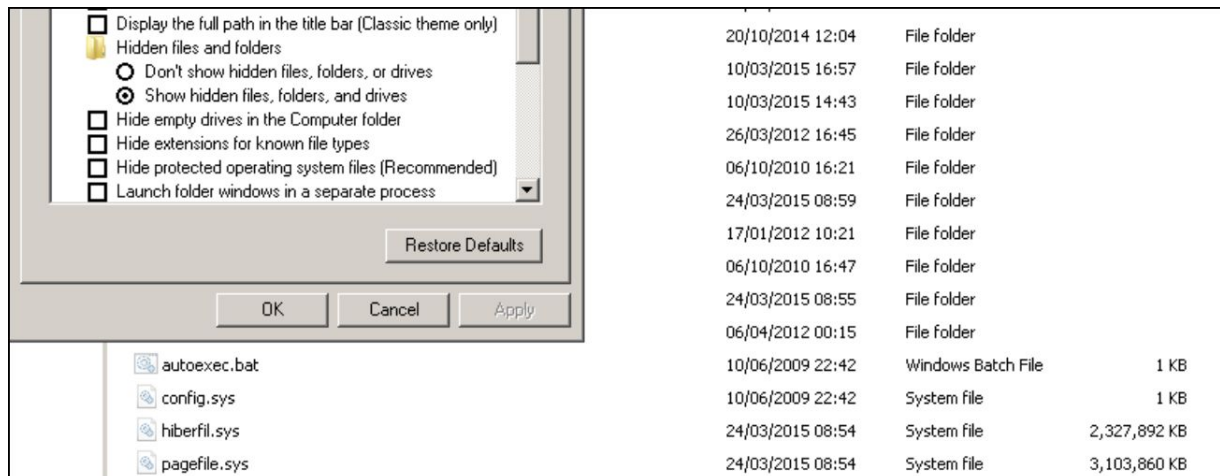


| Image Name          | User Name | CPU | CPU Time | Memory (Private ... | Page Faults | Threads | GDI Objects |
|---------------------|-----------|-----|----------|---------------------|-------------|---------|-------------|
| chrome.exe          | humphrys  | 00  | 00:06:58 | 144,372 K           | 2,001,606   | 41      | 315         |
| realsched.exe       | humphrys  | 00  | 00:00:44 | 168 K               | 1,474,732   | 6       | 4           |
| chrome.exe          | humphrys  | 00  | 00:03:49 | 172,880 K           | 1,148,030   | 16      | 8           |
| chrome.exe          | humphrys  | 00  | 00:04:15 | 55,784 K            | 729,147     | 18      | 143         |
| chrome.exe          | humphrys  | 00  | 00:00:12 | 22,664 K            | 410,698     | 10      | 4           |
| chrome.exe          | humphrys  | 00  | 00:00:17 | 52,868 K            | 328,554     | 11      | 4           |
| chrome.exe          | humphrys  | 00  | 00:00:24 | 155,600 K           | 275,101     | 11      | 4           |
| thunderbird.exe     | humphrys  | 00  | 00:02:16 | 122,940 K           | 240,483     | 34      | 331         |
| chrome.exe          | humphrys  | 00  | 00:00:04 | 7,904 K             | 126,287     | 10      | 4           |
| explorer.exe        | humphrys  | 00  | 00:00:33 | 14,084 K            | 98,955      | 21      | 308         |
| netsession_win.exe  | humphrys  | 00  | 00:00:19 | 4,004 K             | 97,924      | 12      | 4           |
| googledrivesync.exe | humphrys  | 00  | 00:00:37 | 39,316 K            | 91,837      | 29      | 78          |

- Windows 7 Task Manager.
- The above combines both hard and soft page faults.
- "Resource Monitor" (see above) shows hard faults per second.

## Where to swap to?

- Swap to an area of disk dedicated to swapping. Maybe entirely separate to file system.
- On UNIX/Linux, swap to a partition that has no file system on it.
- User doesn't have to see it. Doesn't have file names. Not addressable by user.
- On Windows, swap to the named file `C:\pagefile.sys`
- Might move this onto its own partition.



- Windows: In Folder Options, "Show hidden files" and uncheck "Hide protected OS files".
- Then can see `C:\pagefile.sys` (size here 3 G).

## Hibernation

- Another form of dumping memory to disk is hibernation. Power down with programs open. Write dump of memory to disk. When power up, read image of memory back from disk.
- On Windows, memory image saved in `C:\hiberfil.sys`

## Files

### Files

- File Systems

### Contiguous file allocation

- Files all in one unbroken sequence on the physical disk.
- Problems like with contiguous memory allocation.
- What happens if file grows? Has to be rewritten into larger slot. Takes ages.
- Many files grow slowly,
  - e.g. text editor files.
- But some files grow quickly, unpredictably, or without limit.
  - e.g. HTTP access log, or zip or tar archive file:  

```
tar -cf file.tar $home
```
  - Grows from 0 k to 10 G in a minute or two.

### Non-contiguous allocation

- Like with paging in memory, disk is divided into blocks.
- Blocks are of fixed size, say 1 k or 4 k.
- On Linux there are ways to query block size of different devices at command line.
- On DCU Linux, may not have permission. So here is a quick alternative way:  
Create new file. Will get one block allocated for it:  

```
$ echo 1 > newfile
```

  
See how much space allocated for it (one block).  

```
$ du -h newfile
```

```
4.0K
```
- File is held in a collection of blocks scattered over the disk.
- If file needs more blocks, it can take them from anywhere on disk that blocks are free.

### Index of where the blocks of the file are

- Like pages in memory, blocks can "flow" like liquid into slots around the disk. Don't all need to be in the same place.
- Need some index of where the bits of the file are.
- Various indexing systems using linked lists or tables:
  - File Allocation Table (FAT)
  - NTFS
  - UNIX inode
- See Demonstration of fragmentation (files split into multiple parts).

## Shell script to see blocks allocated to files

```
# compare actual file size with blocks used
```

```
for i in *
do
  ls -ld $i
  du -h $i
done
```

- **Results like (on system with 1 k blocks):**

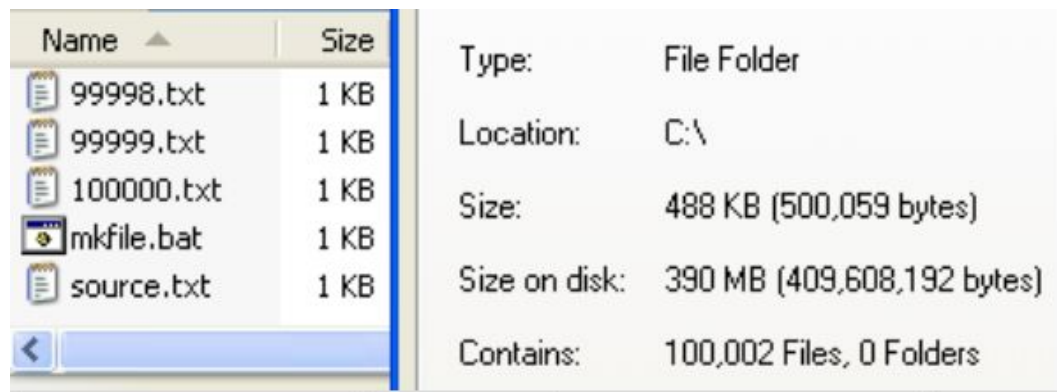
```
-rwxr-xr-x 1 me mygroup 857 Jun 27 2000 synch
1.0K      synch

-rwxr--r-- 1 me mygroup 1202 Oct 25 2013 profile
2.0K      profile

-rwxr-xr-x 1 me mygroup 1636 Oct 28 2009 yo.java
2.0K      yo.java

-rwxr--r-- 1 me mygroup 2089 Oct  8 00:03 flashsince
3.0K      flashsince

-rwxr-xr-x 1 me mygroup 9308 Oct 19 2010 yo.safe
10K       yo.safe
```



| Name       | Size | Type:         | File Folder                |
|------------|------|---------------|----------------------------|
| 99998.txt  | 1 KB | Location:     | C:\                        |
| 99999.txt  | 1 KB | Size:         | 488 KB (500,059 bytes)     |
| 100000.txt | 1 KB | Size on disk: | 390 MB (409,608,192 bytes) |
| mkfile.bat | 1 KB | Contains:     | 100,002 Files, 0 Folders   |
| source.txt | 1 KB |               |                            |

- An extreme experiment to demonstrate wasted space ("slack space") in file systems.
- This person makes 100,000 files of 5 bytes each.
- This is only 500 k of actual data.
- But it needs 400 M of disk space to store it.
- From here.

## Contiguous file allocation is good where possible

- Unlike in memory, where contiguous allocation is dead, in files it has made a bit of a comeback.
- The reason is that there is a lot of disk space and it is cheap, but the speed of read/writing disk has not increased so much - indeed it has got far worse relative to CPU speed.
- To write a contiguous file to disk you don't have to jump the disk head around the disk. You just keep writing after the last location you wrote - the head is in the correct position.
- So modern systems try to use contiguous allocation for small files, only use non-contiguous for large files. To achieve this, they may allocate more space than possibly needed. Some wasted disk space is worth it if it makes disk I/O a lot faster.
- Look up Demonstration of fragmentation (files split into multiple parts) and defragmentation (reducing such splitting and making files contiguous)

### Cache blocks in RAM for speed

- Also to speed things up: OS caches recently-accessed blocks in memory in case needed again. Avoid another disk I/O.

### RAM drive

- RAM drive - RAM formatted with a file system.
- Very fast, but volatile.
- Will be cleared after reboot. Needs to be loaded up with files from disk (non-volatile).
- Need to write changes to files to actual disk or they will be lost.
- Automatic software can take care of the above two.
- Battery-backup RAM drive - survive through power outage.

## Using Files

### Using files

- File - A named section of disk.
- Files implementation: Not necessarily a contiguous section of disk (but that fact may be hidden from users and programs).
- Normally both user and programmer never deal with disk directly, but only by calling named files.
- In some high-performance application (e.g. writing a high-speed search engine), you may need to implement your own file system, but this is obviously difficult and full of dangers.

### File Types

- List of file formats
- Alphabetical list of file extensions
- `file` - query file type
- Programs (machine readable)
  - Extensions: class, exe, com, o, obj, a, dll
  - See also Program File Types
- Program source code (human readable)
  - java, c, cxx, h, hxx, pas, asm
- Programs (human readable) - interpreted scripts
  - js, php, sh, bat, pl
- Program data (machine readable). Often strictly formatted. Precise length of each field pre-defined (for ease of machine reading, and so data can be read into pre-defined fixed-size program variables).
  - Database files.
  - Documents for display. e.g. Word docs (doc), ps, pdf, rtf, tex, dvi
  - Multimedia files - images, audio, video. - gif, jpg, jpeg, mpg, mpeg, ram, avi, qt, au.
- Program data (human readable). Often variable size, free-form text.
  - Preferences files, rc.
  - Documents for display. e.g. HTML docs (htm, html, shtml), xml, Office xml (docx), latex, txt
  - Human readable program data - xml, json
  - See also Human readable program data
- Log files.
- Archive files - tar
- Compressed files - zip, arc, gz, Z.



## File system divisions

- Windows file system can spread over multiple pieces of hardware. Each given its own (single-letter) drive:  
`drive:\dir\file`
- Can also partition a single piece of hardware into multiple drives.
- UNIX file system can spread over multiple pieces of hardware too. But everything appears as sub-directories of a single file hierarchy.
- Path may indicate hardware, something equivalent to:  
`/drive/dir/file`
- or may hide hardware entirely:  
`/dir/file`
- `df`
- Distributed file systems
- Network file systems

## Hierarchical file system

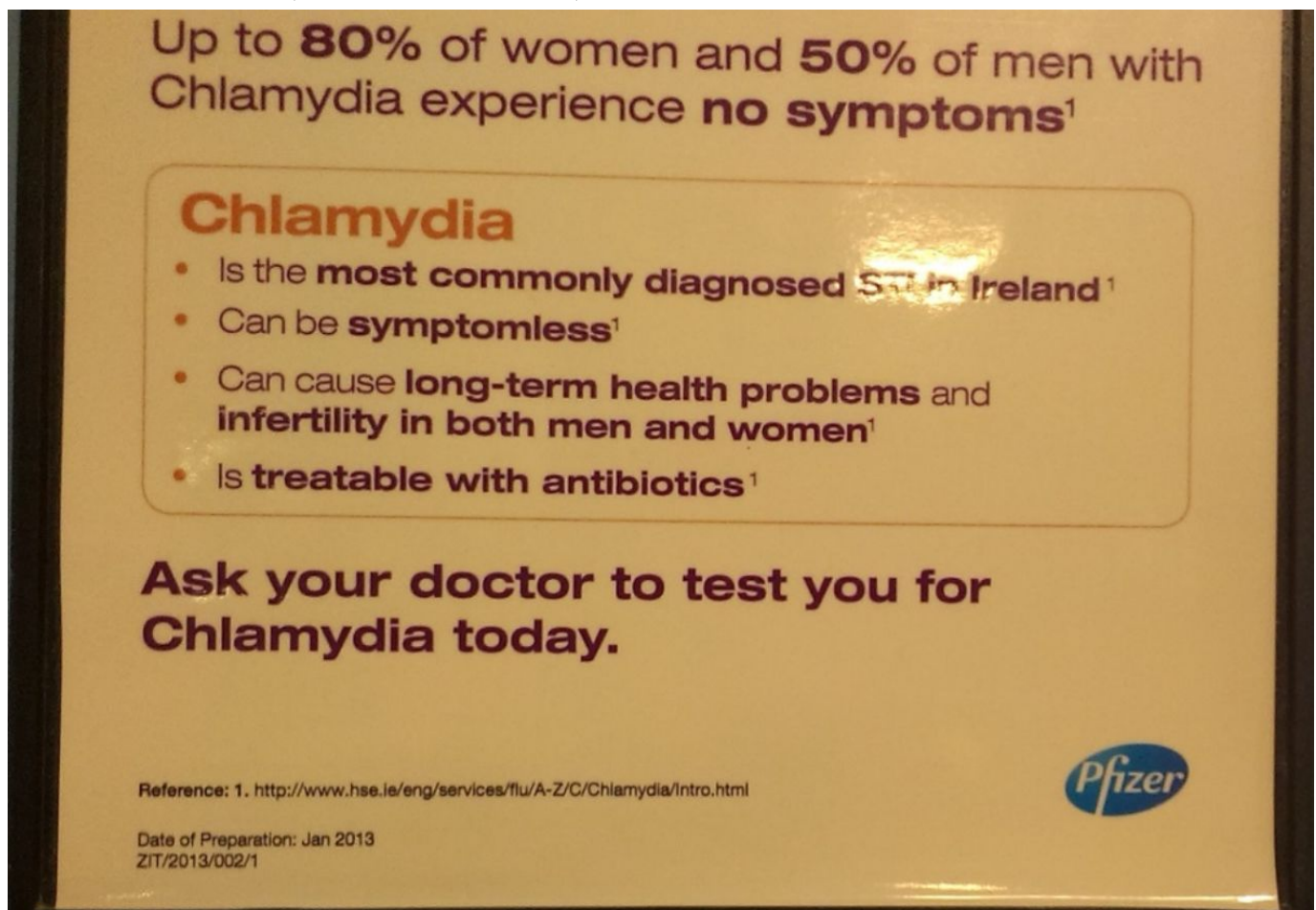
- Can organise files in separate dirs (Many web authors seem not to have discovered sub-dirs!).
- Crucial to keep user files separate from system files (Why?).
- Windows `C:\Users\me`
- UNIX `$HOME`
- Can reuse the same file names in different sub-dirs (like `index.html`).

## Long file names

- All modern OS's allow long filenames:  
`Photos.kenya.apr.1963.html`
- Legacy systems:
  - DOS and Windows before Windows 95 had 8 char name, 3 char extension:  
`phka0463.htm`
- VM/CMS had 8 char filenames and no sub-directories!

### Short file names are good for..

- Short file names are good, though, for:
  1. File names you type. e.g. If you are typing file names at command-line. All-lower-case is easiest to type.
  2. Program names at the command-line (i.e. the program you call has a short filename). sed, grep, ls, cut, etc. All-lower-case is easiest to type.
  3. Some people say also URLs?
- Maybe you should never type URLs. At most you type the host name that you saw somewhere. For everything else you cut and paste, or click.
- Maybe short URLs: <http://en.wikipedia.org/wiki/Othello> make the web a more pleasant experience than long URLs:
  - [http://dmoztools.net/Arts/Literature/World\\_Literature/British/Shakespeare/Works/Plays/Tragedies/Othello/](http://dmoztools.net/Arts/Literature/World_Literature/British/Shakespeare/Works/Plays/Tragedies/Othello/).
- It is nice to have short, "guessable" URLs.
- See "URL as UI"
- See URL shortening. (Used e.g. on Twitter.)



- Short URLs should probably be used in posters and ads:
- This health poster on campus caught my eye.
- This probably should use a shorter, and lowercase, URL, like: [hse.ie/chlamydia](http://hse.ie/chlamydia)

## Symbolic link (cross-link, breaking the hierarchy, "shortcut") in UNIX

- Directory
  - Can selectively break the hierarchy with shortcuts.  
`ln -s dir shortcut`
  - or in Windows see "Create Shortcut"
  - e.g. on one system I used:  

```
$ ls -l /bin
lrwxrwxrwx 1 root      root          9 Apr 14  1997 /bin ->
./usr/bin
```
- File
  - Can also just give a file multiple names:  
`ln -s file secondname`
  - e.g. on DCU Linux:  

```
$ ls -l /bin/ls
lrwxrwxrwx 1 root root 11 Apr  8 21:49 /bin/ls -> /usr/bin/ls

$ ls -l /usr/bin/ps
lrwxrwxrwx 1 root root 7 Feb 12 12:14 /usr/bin/ps -> /bin/ps
```
- Q. Why do programs sometimes call a specific path to a program, e.g. they call /bin/ls rather than just ls ?
  - Can do this on Windows as well (have multiple shortcuts to a data file or program).

## Problems with cross-links

- With shortcuts, if doing a recursive search of disk, can get infinite loop problems, or at least duplication. e.g. List all files on disk. If follow symbolic links may list files twice.
- Q. Also, if delete file, do you delete symbolic link? If so, how do you find them - do you have reverse directory of them? Also, I make symbolic link to other user's file. They delete file. They can't delete my link.
- A. If link doesn't work, so what. Might even leave it dangling as reminder.
- Security
  - If your directory is accessible by others on your local machine, someone on your machine can make it readable by the world on the Web (either maliciously or accidentally):  

```
cd      /homes/your-userid/public_html
ln -s   /homes/other-userid/dir          shortcut
```
  - The world can then read other user's directory through:  
`http://host/~your-userid/shortcut/`
  - Has valid uses too. Might want to make one of your own dirs visible without having to have it under public\_html, e.g. public\_html disk is full, dir is on another disk.
  - Another example - SAMBA or read-write ftp may only drop you in home directory rather than root directory and you may not be able to go upwards. What you do is put symbolic links in your home directory and you can access any directory through them:  

```
ln -s /var/mail  email
ln -s /htdocs    ht
```

### "Hierarchy with some cross-links" a very powerful model

- General conclusion is that a basic hierarchy, with some cross-links for difficult points, is excellent way to structure complex data (e.g. Open Directory) - rather than total cross-link free-for-all on one hand (e.g. the Web with just search engines and no directories), or rigid hierarchy on other (e.g. Dewey library system).
- Interestingly, family trees are also basically hierarchical, with arbitrary cross-links, rather than strictly hierarchical as many people seem to think.

### Recycle bin (Windows)

- Windows Recycle bin visible through GUI, but also visible as directory through Windows command line:
  - `cd c:\$recycle.bin`
  - `dir` (apparently empty)
  - `dir /ah` (show hidden files)

### Backup

- If it's data (1's and 0's), there's no real excuse for losing it. You can make automated copies and store them all over the world. Disk space is big and cheap. Machines are often idle. The network is always on. Backups can be automated across the network by scripts.
- In future, backup and long-term storage will be increasingly important service, like a bank.
- Removable media - DVDs, CDs, tapes, USB keys, external hard disk.
- v.
- Backup to cloud / server. Distributed file system. Network read-write ftp, automated scripts, mirrors.
  
- List of backup software
- Comparison of backup software
- Online backup services
- Comparison of online backup services

### Other people back you up

- Even if you back up nothing, your web pages are being backed up by other people:
- Google cache (click on "Cached")
- Microsoft cache (click on "Cached page")
- Internet Archive
- Flickr - network photo storage
- Other social media.
- Often our data is on a remote backed-up server by default

## Backup policy

1. Periodically dump entire file system to backup.  
v.
  2. Keep a running "mirror", and only backup things that have changed since last time they were synch-ed.
- Perhaps only backup user files.
  - OS, system and application files can be recovered from install CDs / tapes.
  - Which of these is the most dangerous:
    1. Keep 1 synchronised copy of your files. Backup the changes every night.
    2. Keep 1 synchronised copy of your files. Backup the changes every hour.
    3. Take a copy of all of your files once a week. Keep all these old copies. Do no backups at all during the week.
    4. Take a copy of all of your files once a month. Keep all these old copies. Do no backups at all during the month.
  - Remember - it may take days or even months before an intrusion and destruction, or accidental damage, is noticed.
  - User may realise 2 years later that he has deleted some file and needs it back.
  - VAX/VMS (DEC) could be set to keep all drafts of a file since created.
  - The equivalent of `ls` would hide all except the latest one by default. Unless explicitly asked otherwise.
  - Programming with DCL would work with the latest one by default. Unless explicitly asked otherwise.
  - Lot to be said for such an approach, now that disk space is cheap.
  - Versioning file systems
  - Google Docs saves all old drafts/versions of docs.

## Backup policy

1. Periodically dump entire file system to backup.  
v.
  2. Keep a running "mirror", and only backup things that have changed since last time they were synch-ed.
- Perhaps only backup user files.
  - OS, system and application files can be recovered from install CDs / tapes.
  - Which of these is the most dangerous:
    1. Keep 1 synchronised copy of your files. Backup the changes every night.
    2. Keep 1 synchronised copy of your files. Backup the changes every hour.
    3. Take a copy of all of your files once a week. Keep all these old copies. Do no backups at all during the week.
    4. Take a copy of all of your files once a month. Keep all these old copies. Do no backups at all during the month.
  - Remember - it may take days or even months before an intrusion and destruction, or accidental damage, is noticed.
  - User may realise 2 years later that he has deleted some file and needs it back.
  - VAX/VMS (DEC) could be set to keep all drafts of a file since created.
  - The equivalent of `ls` would hide all except the latest one by default. Unless explicitly asked otherwise.
  - Programming with DCL would work with the latest one by default. Unless explicitly asked otherwise.
  - Lot to be said for such an approach, now that disk space is cheap.
  - Versioning file systems
  - Google Docs saves all old drafts/versions of docs.