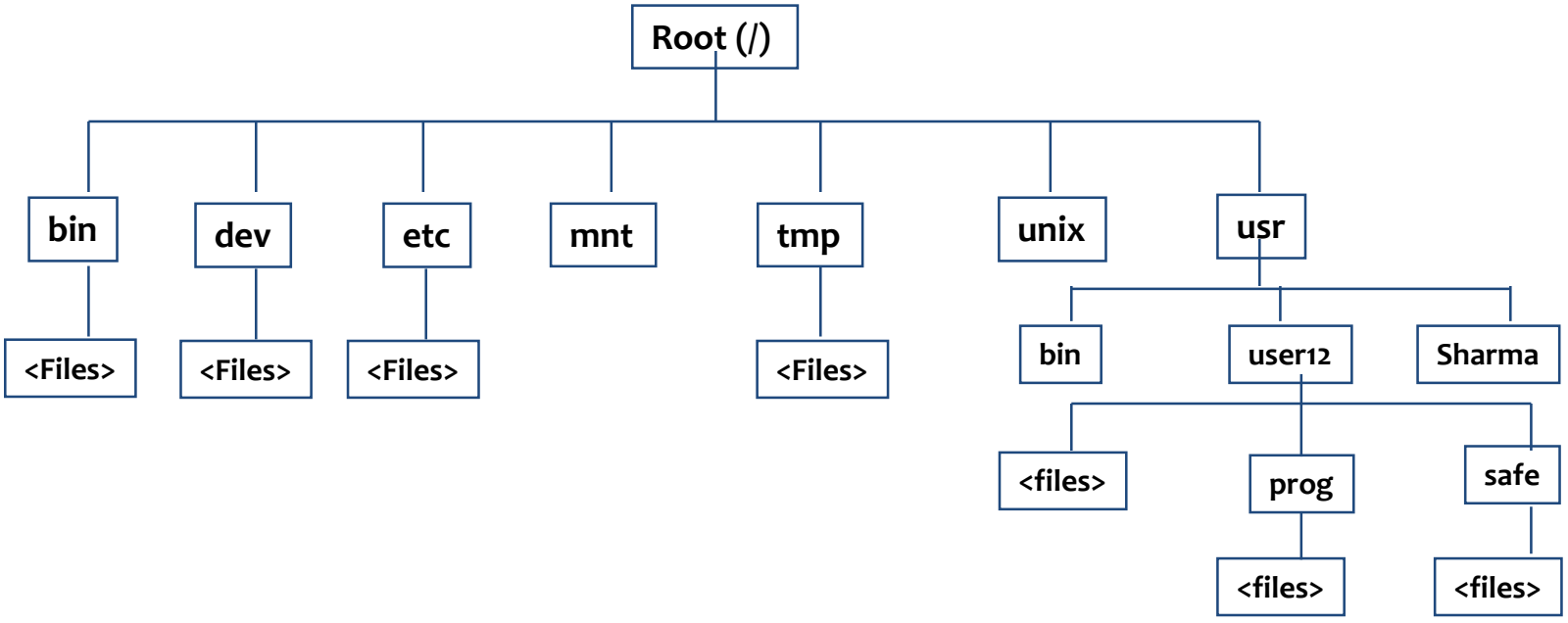
UNIX

UNIX File System

File System Structure



File System Structure

- / bin **: commonly used UNIX Commands like who, ls**
- /usr/bin **: cat, wc etc. are stored here**
- /dev **: contains device files of all hardware devices**
- /etc **: contains those utilities mostly used by system administrator**
  - Example: passwd, chmod, chown

File System

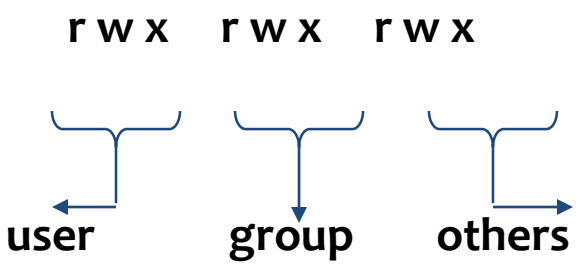
- /tmp **: used by some UNIX utilities especially vi and by user to store temporary files**
- /usr **: contains all the files created by user, including login directory**
- /unix **: kernel**
- **Release V:**
  - It does not contain / bin.
  - It contains / home instead of /usr.

File Types in UNIX

- **We have the following file types in UNIX:**
  - Regular File
  - Directory File
  - Device File
  - Pipe file
  - Link file

File Permissions in UNIX

- **File Access Permissions**



File Permissions in UNIX

- **Permissions are associated with every file, and are useful for security.**
- **There are three categories of users:**
  - Owner (u)
  - Group (g)
  - Others (o)
- **There are three types of "access permissions":**
  - Read (r)
  - Write (w)
  - Execute (e)

pwd Command

- **The** pwd **command checks current directory.**

```
$ pwd
```

  - **Output:** /usr/user12

cd Command

- **The** cd **command changes directories to specified directory**
- **The directory name can be specified by using absolute path (Full Path) or relative path**

```
$ pwd
```

- **Output:** /usr/user12

```
$ cd Prog
$ pwd
```

- **Output:** /usr/user12/Prog

cd Command

- **Moving one level up:**

```
$ cd ..
```

Switching to home directory:

```
$ cd
```

- **Switching to /usr/sharma:**

```
$ cd  /usr/Sharma
```

- **Switching to root directory:**

```
$ cd /
```

logname Command

- **The** logname **command checks the login directory.**

```
$ logname
```

Output: user12

ls Command

- **The ls command lists the directory contents.**
- **Example:**

```
$ ls
```

Output:

```
a.out
pack1
pack2
test
test.c
```

ls Command

- **Options available in ls command:**

Option	Description
-x	**Displays multi columnar output (prior to Release 4)**
-F	**Marks executables with *and directories with /**
-r	**Sorts files in reverse order (ASCII collating sequence by default)**
-l	**The long listing showing seven attributes of a file**
-d	**Forces listing of a directory**
-a	**Shows all files including ., .. And those beginning with a dot**

ls Command

- **Options available in ls command:**

Option	Description
-t	**Sorts files by modification time**
-R	**Recursive listing of all files in sub-directories**
-u	**Sorts files by access time (when used with the –t option)**
-i	**Shows i-node number of a file**
-s	**Displays number of blocks used by a file**

ls Command

- **Example:**

```
$ ls – l
```

  - It displays output as follows which includes 7 columns total 8:

```
-rw-rw-rw-    1 user12 group 44   May 9 09:08 dept.h
-rw-rw-rw-    1 user12 group 212  May 9 09:08 dept.q
-rw-rw-rw-    1 user12 group 154  May 9 09:08 emp.h
```

ls Command

- **Consider the first column:**

  Field1 --> mode

  - r w x r w x r w x

    □ □ □

    □ --> user permissions

    □ --> group permissions

    □ --> others permissions

ls Command

- **File type**
  - 1 st character represents file type:

    - r w x r w x r w x

    - - --> regular file
    - d --> directory file
    - c --> character - read
    - b --> block read

ls Command

- **Field2 : indicates number of links**
- **Field3 : File owner id**
- **Field4 : Group id**
- **Field5 : File size in bytes**
- **Field6 : Date/time last altered**
- **Field7 : Filename**

cat Command

- **The** cat **command is used for displaying and creating files.**
  - To display file:

```
$ cat dept.lst
```

```
01|accounts|6213
02|admin|5423
:
06|training|1006
```

  - To create a file:

```
$cat > myfile
```

    - This is a new file
    - Press ctrl-d to save the contents in file myfile

cat Command

- **The** cat **command can be used to display contents of more than one file.**
  - It displays contents of pack2 immediately after displaying pack1.

```
$ cat pack1 pack2
```

Input and Output Redirection

- **Standard Input: Keyboard**
- **Standard Output : Monitor**
- **Standard Error : Monitor**

- **Redirection operators:**
  - < : Input Redirection
  - > : Output Redirection
  - 2> : Error Redirection
  - >> : Append Redirection

Redirection

- Input redirection: **Instead of accepting i/p from standard i/p(keyboard) we can change it to file.**
  - **Example:** $cat < myfile will work same as $cat myfile
  - < indicates, take i/p form myfile and display o/p on standard o/p device.
- Output redirection: **To redirect o/p to some file use >**
  - **Example:** $cat < myfile > newfile
  - The above command will take i/p from myfile and redirect o/p to new file instead of standard o/p (monitor).

Redirection

- $ cat < file1.txt > result is same as $cat file1.txt > result.

```
$ cat result
```

Output: 2 15 20

- >> is append redirection
- The given command will append the contents of file1.lst in result file.

```
$ cat  < file1.lst >> result
$ cat result
```

Output: 2 15 20
4 4 8

cat file exist/not exist

- **Consider an example of cat –(file exist/not exist):**

```
$ cat abc.txt > pqr.txt 2> errfile.txt
```

  - If file abc.txt exists:
    - Then contents of the file will be sent to pqr.txt. Since no error has occurred nothing will be transferred to errfile.txt.
  - If abc.txt file does not exist:
    - Then the error message will be transferred to errfile.txt and pqr.txt will remain empty.

cp Command (copy file)

- **The** cp **(copy file) command copies a file or group of files.**
- **The following example copies file pack1 as pack2 in test directory.**
  - Example:

```
$ cp pack1 temp/pack2
Option - i (interactive)
      $cp  - i  pack1    pack2
      cp: overwrite pack2          ? y
Option -r (recursive) to copy entire directory
$cp  -  r   temp   mytemp
```

rm Command (delete file)

- **The** rm **(remove file) command is used to delete files:**

```
$ rm pack1    pack2     pack3
      $ rm   *
        Are you sure? y
Option - i (interactive delete)
      $ rm  -  i  pack1     pack2
      pack1 : ? Y
      pack2  :?  Y
Option   -   r (recursive delete)    (Avoid using this option)
```

mv Command

- **The** mv **command is used to rename file or group of files as well as directories.**

```
$ mv   pack1  man1
```

- **The destination file, if existing, gets overwritten:**
  - Example: $ mv temp doc
  - Example: $ mv pack1 pack2 pack3 man1
    - It will move pack1, pack2 & pack3 to man1 directory

wc Command

- **The** wc **command counts lines, words, and character depending on option.**
- **It takes one or more filename as arguments.**
- **no filename is given or - will accept data from standard i/p.**

```
$ wc myfile
3  20  103 myfile
$wc       or    $wc  -
This is standard input
press ctrl-z to stop
```

- **Output:** 2 8 44

wc Command

```
$ wc  infile test
```

Output:

3	20	103	infile
10	100	180	test
13	120	283	total

```
$ wc  -  l  infile
```

Output: 3 infile

```
$ wc  -  wl  infile
```

Output: 20 3 infile

The following command will take i/p from infile and send o/p to result file

```
$ wc < infile > result
$ cat result
```

Output: 2 12 60

cmp Command

- **cmp Command:**

```
$ cmp file1.txt file2.txt
file1.txt file2.txt differ: char 41, line 2
$ cmp file1.txt file1.txt
```

comm Command

- **comm Command:**
  - The comm command compares two sorted files. It gives a 3 columnar output:
    - First column contains lines unique to the first file.
    - Second column contains lines unique to the second file.
    - Third column displays the common lines.

comm Command

```
$ cat cfile1.lst
A
p
K
X
```

```
$ cat cfile2.lst
A
F
K
W
X
Z
```

```
$ comm cfile1.lst cfile2.lst
                A
        F
p
                K
        W
                 X
        Z
```

```
$ comm -12 cfile1.lst cfile2.lst
A
K
X
```

diff Command

- **The** diff **command is used to display the file differences. It tells the lines of one file that need to be changed to make the two files identical.**
  - Example:

```
$ diff cfile1.lst cfile2.lst
2c2
< p
> F
3a4
> W
4a6
> Z
```

tr Command

- **The** tr **command accepts i/p from standard input.**
- **This command takes two arguments which specify two character sets.**
- **The first character set is replaced by the equivalent member in the second character set.**
- **The –s option is used to squeeze several occurrences of a character to one character.**

tr Command

- Example 1: **To squeeze number of spaces by single space:**

```
$ tr –s “ ”  < file1.txt
```

- Example 2: **To convert small case into capital case:**

```
$ tr “[a-z]” “[A-Z]” < file1.txt
ONE
TWO
THREE
FOUR
```

more Command

- **The** more **command, from the University of California, Berkeley, is a paging tool.**
- **The** more **command is used to view one page at a time. It is particularly useful for viewing large files.**
- **Syntax for** more **command is as follows:**

```
more <options> <+linenumber> <+/pattern> <filename(s)>
```

- Example: **To display file1.txt one screenful at a time**

```
$ more file1.txt
```

chmod Command (Alter File Permissions)

- **The** chmod **command is used to alter file permissions:**
- **Syntax:**

```
chmod <category> <operation> <permission> <filenames>
```

Category	Operations	Attribute
u-user	+assigns permission	r-read
g-group	-remove permission	w-write
o-others	=assigns absolute permission	x-execute
a-all		

chmod Command (Alter File Permissions)

- **Example 1:**

```
$ chmod u+x note
$ ls - l note
      -rwx r-- r --1 ……      note
```

- **Example 2:**

```
$ chmod ugo+x          note
$ ls - l note
  -rwxr-xr-x ……       note
```

  - When we use + symbol, the previous permissions will be retained and new permissions will be added.
  - When we use = symbol, previous permissions will be overwritten.

chmod Command (Alter File Permissions)

- **Example 3:**

```
$ chmod u-x, go+r        note
      $ chmod u+x        note      note1      note2
      $ chmod o+wx       note
      $ chmod ugo=r      note
```

chmod Command (Alter File Permissions)

- **Octal notation:**
  - It describes both category and permission.
  - It is similar to = operator (absolute assignment).
    - read permission: assigned value is 4
    - write permission: assigned value is 2
    - execute permission: assigned value is 1
  - Example 1:

```
$ chmod   666 note
```

    - It will assign read and write permission to all.

chmod Command (Alter File Permissions)

- Example 2:

```
$ chmod  777 note
```

  - It will assign all permissions to all.

- Example 3:

```
$ chmod  753 note
```

mkdir Command

- **The** mkdir **command creates a directory.**

```
$ mkdir mytemp
```

```
$ mkdir dir1     dir1/example          dir1/data
```

```
$ mkdir          dir1/example          doc
```

  - It will give error - Order important.

rmdir Command

- **The** rmdir **command is used to remove directory.**
- **Only empty dir can be deleted.**
- **More than one dir can be deleted in a single command.**
- **Command should be executed from at least one level above in the hierarchy.**

rmdir Command

```
$ rmdir doc
```

```
$ rmdir doc/example doc
```

```
$ rmdir doc  doc/example
```

- It will give error.

Internal and External Commands:

- External commands
  - A new process will be set up
  - The file for external command should be available in BIN directory
  - E.g – cat, ls , Shell scripts
- Internal commands
  - shell's own built in statements, and commands
  - No process is set up for such commands.
  - E.g cd , echo

UNIX

Filters

What is a Filter?

- **Filters are central tools of the UNIX tool kit.**
- **Commands work as follows:**
  - Accept some data as input.
  - Perform some manipulation on the inputted data.
  - Produce some output.
- **Most of them work on set of records, with each field of a record delimited by a suitable delimiter.**
- **When used in combination, they can perform complex tasks too.**

head Command

- **The** head **command, by default, will display the first 10 lines of a file.**
  - **Example 1:** To display fist 10 lines from file books:

```
$head books
```

  - **Example 2:** To display first 5 lines from file books:

```
$head –5 books
```

- **Single command can be used to display lines from more than one file.**

```
$ head -1 PuneEmp MumbaiEmp
```

tail Command

- **The** tail **command is useful to display last few lines or characters of the file.**
  - **Example 1:** To display last ten lines from books:

```
$tail books
```

  - **Example 2:** To display last seven lines:

```
$tail –7 books
```

  - **Example 3:** To display lines from the 10th line till end of the file:

```
$tail +10 books
```

  - **Example 4:** To display last 5 characters of the file:

```
$tail -5c books
```

cut Command

- **The** cut **command retrieves selected fields from a file.**

```
$ cut [options] <filename>
```

  - Options :
    - -c : selects columns specified by list
    - -f : selects fields specified by list
    - -d : field delimiter (default is tab)

cut Command

- **Example 1:** To display 2nd and 3rd field from file bookDetails.lst:

```
$ cut -d"|" -f2,3 bookDetails.lst
```

- **Example 2:** To display characters from 1st to 4th and 31st to 35th from file bookDetails.lst :

```
$ cut -c1-4,45-50 bookDetails.lst
```

paste Command

- **The paste command is used for horizontal merging of files.**

```
$paste <file1><file2><Enter>
```

  - Options : -d (Field delimiter)
  - **Example 1:** To paste enum.lst and ename.lst files:

```
$ paste enum.lst ename.lst
```

  - **Example 2:** To paste enum.lst and ename.lst files with '|' character as delimiter:

```
$ paste –d'I' enum.lst ename.lst
```

sort Command

- **The** sort **command is useful to sort file in ascending order.**

```
$sort <filename>
```

  - Options are:
    - -r : Reverse order
    - -n : Numeric sort
    - -f : Omit the difference between Upper and lower case alphabets
    - -t : Specify delimiter
    - -k : to specify fields as primary or secondary key
  - Example:

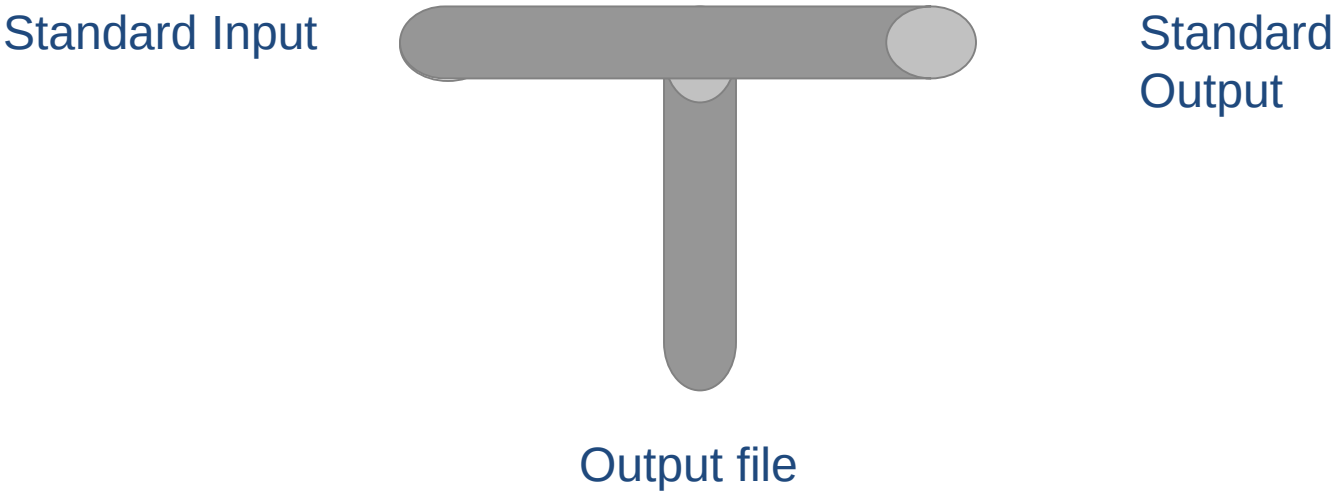
```
$ sort -t"|" +1 bookDetails.lst
$sort –k3,3  -k2,2 books
```

uniq Command

- **The** uniq **command fetches only one copy of redundant records and writes the same to standard output.**
  - –u option: It selects only non-repeated lines.
  - –d option: It selects only one copy of repeated line.
  - -c option: It gives a count of occurrences.
- **To find unique values, the file has to be sorted on that field.**
  - **Example:** To find unique values from file myfile.lst

```
$ uniq myfile.lst
```

tee Command



- To display contents of file books on screen as well as save it in the file:

```
$ tee user.txt < books
```

find Command

- **The** find **command locates files.**

```
find <path list>  <selection criteria> <action>
```

  - To find the file named **.profile** starting at the root directory in the system **-print** specify the action:

```
$ find / -name .profile –print
```

  - To find the file named myfile starting at the root directory in the system

```
find / -type f -name "myfile" -print
```

grep Command

- **The syntax for** grep **command is as follows:**

```
grep <options> <pattern> <filename(s)>
```

  - The following example will search for the string Unix in the file **books.lst**. The lines which match the pattern will be displayed.

```
grep ‘Linux' books.lst
```

grep Command

- **Options of grep:**
  - c : It displays count of lines which match the pattern.
  - n : It displays lines with the number of the line in the text file which match the pattern.
  - v : It displays all lines which do not match pattern.
  - i : It ignores case while matching pattern.
  - -w : It forces grep to select only those lines containing matches that form whole words

grep Command

- To print all lines containing “rose” regardless of case:

```
$grep -i mogra flower.txt
```

- To print all lines containing “rose” as a word:

```
$grep -w mogara flower.txt
```

- To print all lines not containing “rose”:

```
$grep -v mogara flower.txt
```

grep Command

- **Regular Expression:**

Expression	Description
^ (Caret)	match expression at the start of a line, as in ^A.
$ (Question)	match expression at the end of a line, as in A$.
\ (Back Slash)	turn off the special meaning of the next character, as in \^.
[ ] (Brackets)	match any one of the enclosed characters, as in [aeiou]. Use Hyphen "-" for a range, as in [0-9].
[^]	match any one character except those enclosed in [ ], as in [^0-9].
. (Period)	match a single character of any value, except end of line.
* (Asterisk)	match zero or more of the preceding character or expression.
\{x,y\}	match x to y occurrences of the preceding.
\{x\}	match exactly x occurrences of the preceding.
\{x,\}	match x or more occurrences of the preceding.

grep Command

- **Examples of Regular Expression:**

Example	Description
grep "smile" files	**search *files* for lines with 'smile'**
grep '^smile' files	**'smile' at the start of a line**
grep 'smile$' files	**'smile' at the end of a line**
grep '^smile$' files	**lines containing only 'smile'**
grep '\^s' files	**lines starting with '^s', "\" escapes the ^**
grep '[Ss]mile' files	**search for 'Smile' or 'smile'**
grep 'B[oO][bB]' files	**search for BOB, Bob, BOb or BoB**
grep '^$' files	**search for blank lines**
grep '[0-9][0-9]' file	**search for pairs of numeric digits**

fgrep Command

- **The fgrep command is similar to grep command.**
- **Syntax:**

```
$fgrep  [ -e pattern_list] [-f pattern-file] [pattern] [Search file]
```

- **The** fgrep **command is useful to search files for one or more patterns, which cannot be combined together.**
- **It does not use regular expressions. Instead, it does direct string comparison to find matching lines of text in the input.**

fgrep Command

- **Options of fgrep command:**
  - -e pattern_list :
    - It searches for a string in pattern-list.
  - -f pattern-file :
    - It takes the list of patterns from pattern-file.
  - pattern
    - It specifies a pattern to be used during the search for input.
    - It is same as grep command.
  - E.g To search books file for all patterns stored in mypattern file

    $ fgrep –f mypattern books.lst

egrep Command

- **The** egrep **command works in a similar way. However, it uses extended regular expression matching.**
  - Syntax:

```
egrep [ -e pattern_list ] [-f file ] [ strings ] [ file]
```

  - **Example:** To find all lines with name “aggrawal” even though it is spelled differently:

```
$ egrep '[aA]gg?[ar]+wal' stud.lst
```

Linux Internals

Linux Process Management

Contents

- **This module covers**
  - What is Process?
  - Process Descriptor
  - The current macro
  - The process list
  - Process switching
  - Process Creation and Termination
  - Process Scheduling
  - Threads
  - Threads Vs Process
  - Signals
  - Sending and Caching Signals

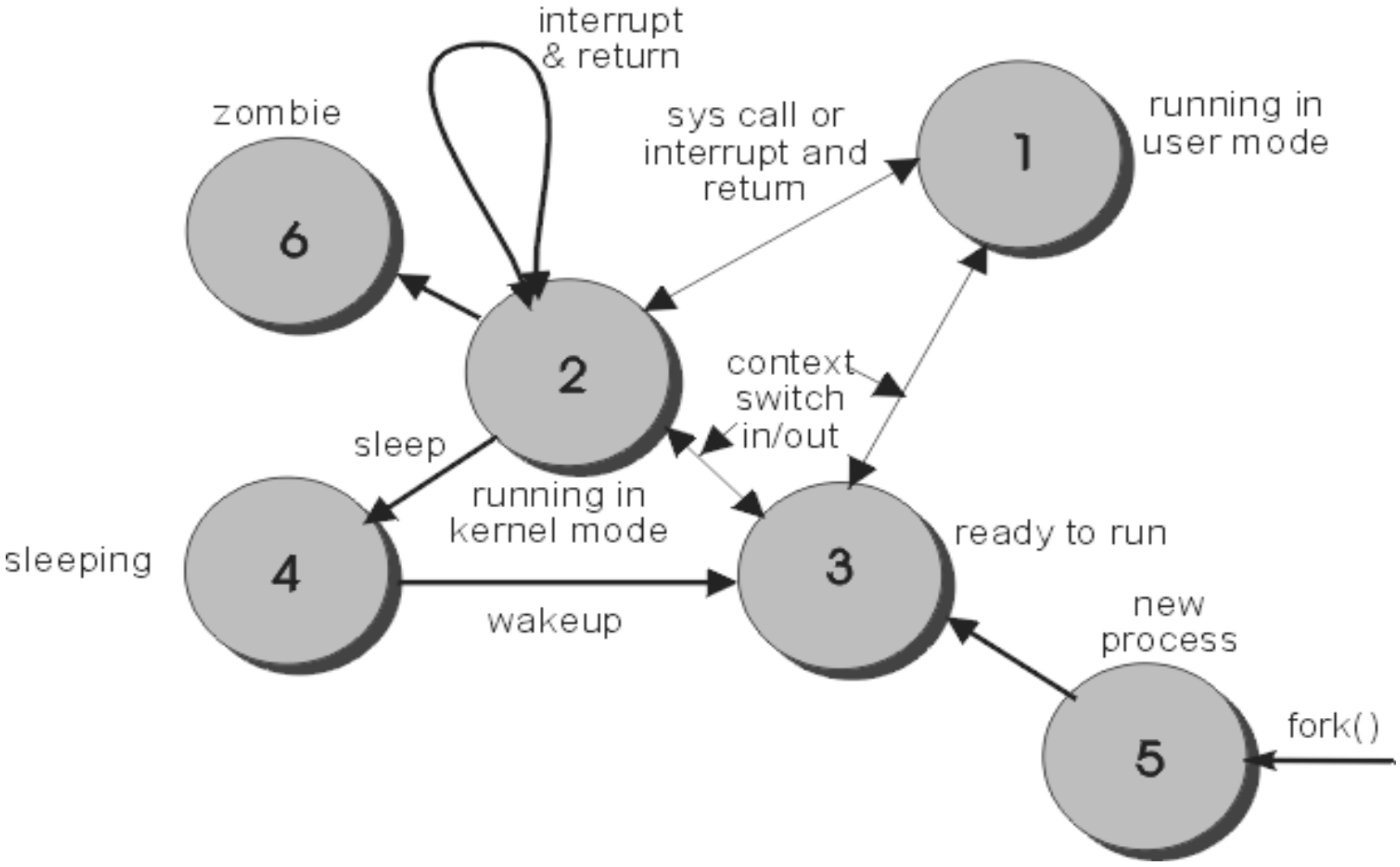
What is Process?

- A process is usually defined as an instance of a program in execution; thus, if 16 users are running vi at once, there are 16 separate processes (although they can share the same executable code)
- Processes are often called "tasks" in Linux source code
- Process include the program code in the text section , data section containing the global variables, a set of resources such as open files and pending signals and one or more threads of execution

Process context

- **The context of a process is all of the characteristics, settings, values, etc., that a particular program uses as it runs, as well as those that it *needs* to run**
- **Even the internal state of the CPU and the contents of all its registers are part of the context of the process**
- **When a process has finished having its turn on the CPU and another process gets to run, the act of changing from one process to another is called a *context switch***

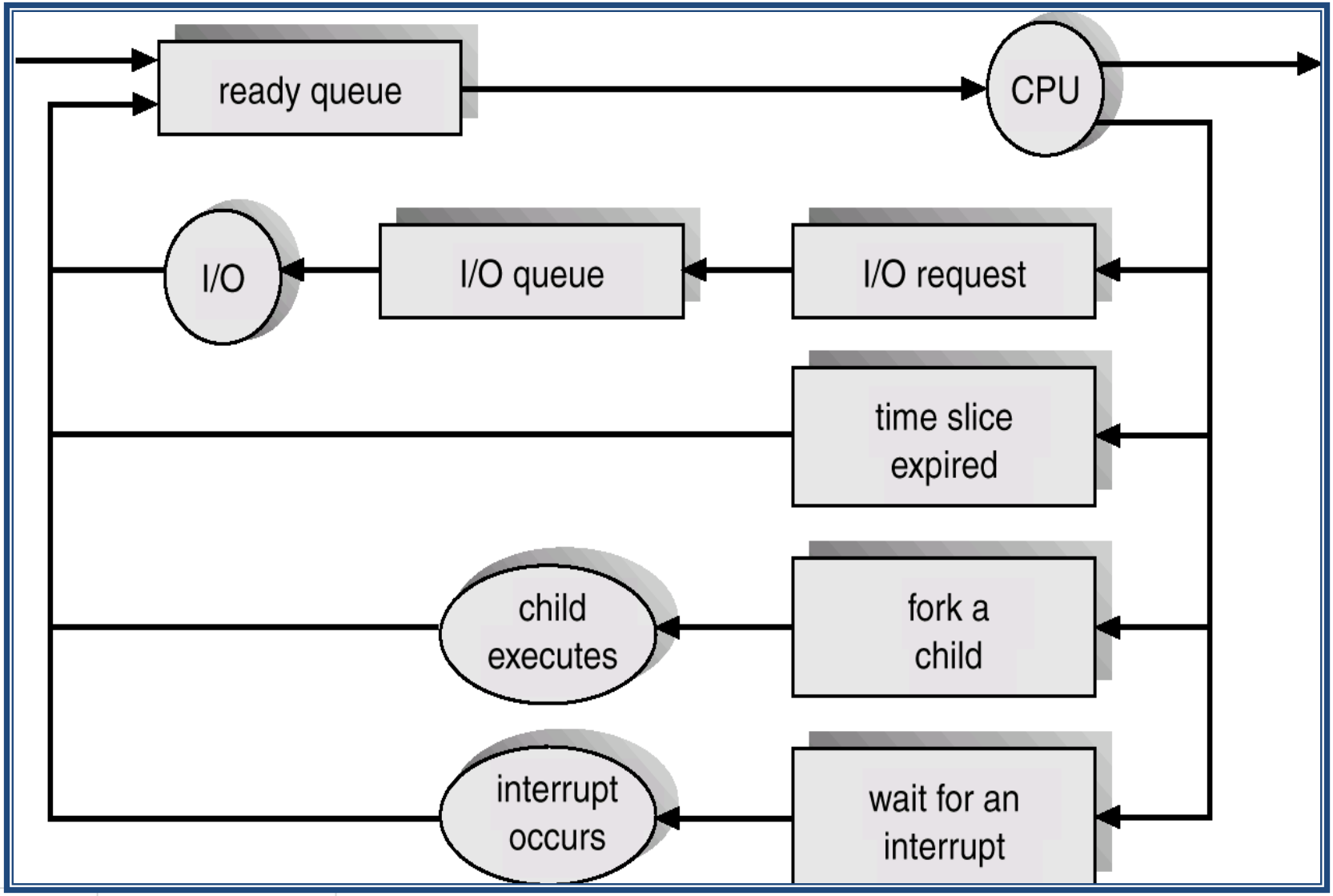
The Life Cycle of Processes



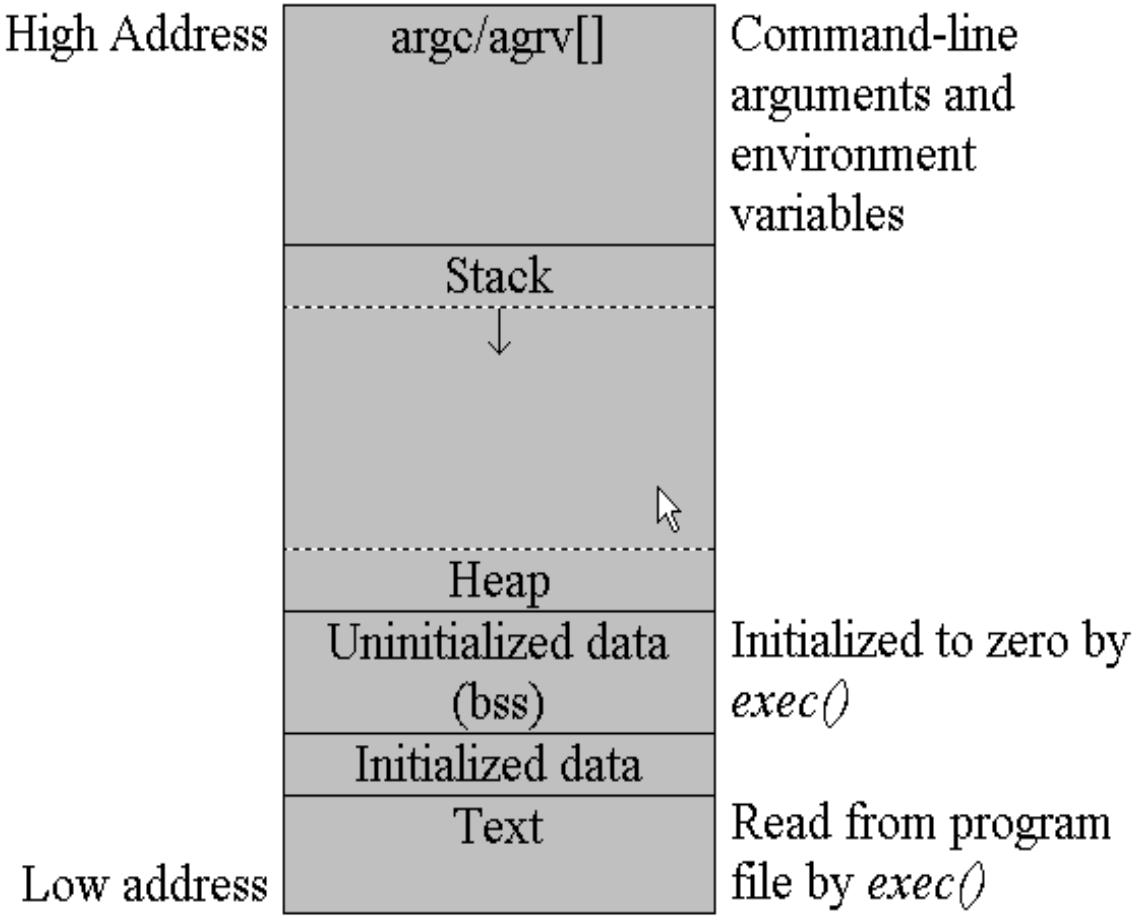
Process Scheduling

- **The set of rules used to determine when and how selecting a new process to run is called *scheduling policy***
- **Linux scheduling is based on the *time-sharing* technique**
  - Several processes are allowed to run "concurrently," which means that the CPU time is roughly divided into "slices," one for each runnable process.Of course, a single processor can run only one process at any given instant. If a currently running process is not terminated when its time slice or *quantum* expires, a process switch may take place.
- **Time-sharing relies on timer interrupts and is thus transparent to processes. No additional code needs to be inserted in the programs in order to ensure CPU time-sharing**

Queuing-Diagram of Process Scheduling



Typical Memory layout of process



Typical logical memory layout of a process