Unix System Programing

Basic Commands..

sourav@sourav-Lenovo-ideapad-310-15IKB:~$ date
Wed Dec 27 18:39:16 IST 2017

sourav@sourav-Lenovo-ideapad-310-15IKB:~$ who
sourav tty7 2017-12-27 21:23 (:0) //'tty' stands for 'teletype' synonym for terminal

sourav@sourav-Lenovo-ideapad-310-15IKB:~$whom
Whom : not found

CTL+U : For line kill, **CTL+D** : For stopping input from source

Manual Page Entry..

- The Unix programmers manual describes most of what you need to know about the system

- To see the description of a system function:

  man system_fn_name

- Example:

  $ man who
  $ man printf
  $ man man

Create a File..

```
$ ed                  //invokes text editor
   a                  // Append
Now type
 .                    //stop the append
w junk                //write into file junk
 9                    //Number of characters append
Q                     //Quit
$
```

Modifying the same created file:

```
$ ed  junk
     a                // Append
……..
……..

$
```

ls Command..

- ls command list the file names in a directory
- ls command not show the content of files
- Example:
  - Suppose in a directory there are two files ‘temp’ and ‘junk’
  - $ ls
    junk
    temp
    $

```
ls filenames //list only the named file
ls –t          // list in time order, most recent first
ls –l          // list long, more information
ls –u          // list by time last used
```

Printing File Contents..

- cat and pr command show the printing file content
- Example:
  - $ cat junk
    how are you?
    $
- pr command also print the contents of a file but with file creation date and added some blank line
- Example:
  - $ pr junk
    sep 26 16:25 2017 junk page 1

    how are you?
    (60 more blank lines)

    $

Rename, Copying, Removing File

- mv command renaming a file from one name to another
- Example:
  - $ mv junk temp
    $
- cp command copy a file to another file
  - Example:
    - $ cp temp temp1
      $
- rm command removes file you named
- Example:
  - $ rm temp
    $

Word Counting and Pattern Match

- wc command counts the lines, words and characters in a file
- Example:
  - $ wc poem
    8 46 263 poem
    $
- grep command searches files for lines that match a pattern
  - Example:
    - $ grep fly poem
      fly like a bird
      my dream is to fly high
      $
- grep command also look for lines that don't match with pattern
  - Example:
    - $ grep -v fly poem
      sky is blue
      Birds have wings
      $

Sorting

- sort command sorts its input in alphabetical order line by line
- Example:
  - $ sort temp

```
an apple
how are you
there are so many persons
$
```

```
sort –r    //sort in reverse order
sort –n    //sort in numeric order
sort –nr   //sort in reverse numeric order
```

- tail command prints number of last line you mentioned
  - $ tail -1 temp

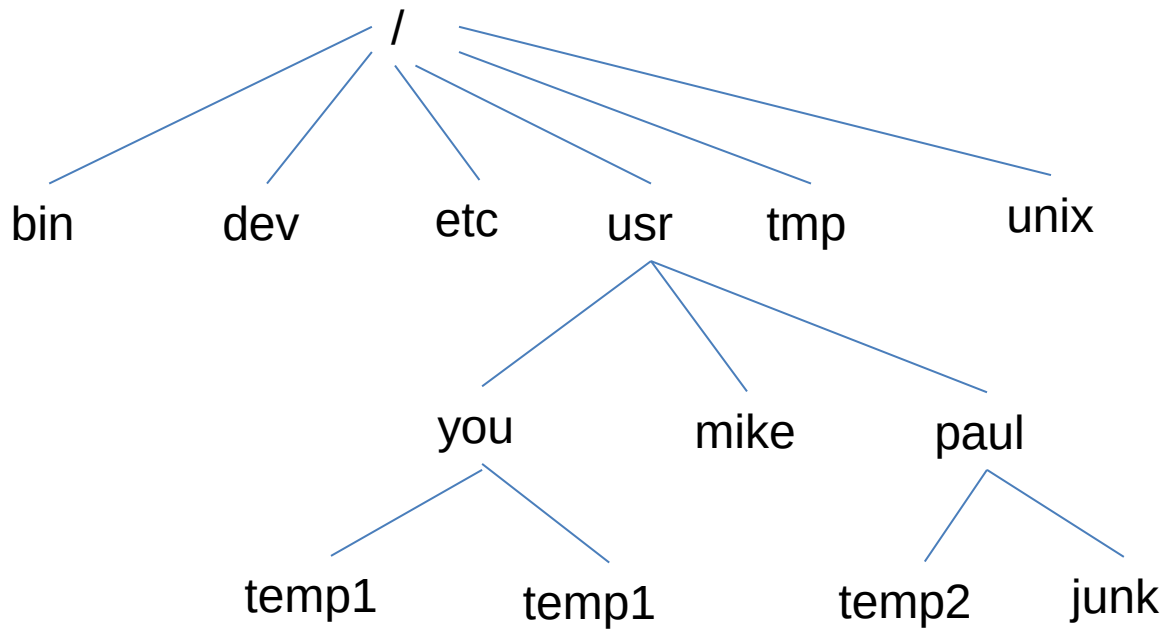
```
there are so many persons
$
```

Compare Two Files

- cmp command find the line number where two files different
- Example:
  - $ cmp temp1 temp2
    temp1 temp2 differ: line 2 char 8
    $

- diff command reports the lines that are different
  - Example:
    - $ diff temp1 temp2
      how are you?
      how are u?
      $

Directories



Directories

- cat /usr/you/temp1
- ls /usr/paul
- cp junk /usr/paul/junk
- ed /usr/you/temp1
- Change directories
  - cd /usr/paul
- Make directories
  - mkdir book
- Show the current directory
  - pwd
- Remove a directory
  - rmdir book

The Shell

- Shell is a command interpreter between you and kernel
- The benefits of shell
  - Filename Shorthands : You can pick up a whole set of file names by specifying a pattern– the shell will find the file names that match your pattern
  - Input-Output redirection: You can arrange input to come from a file instead going to terminal, similarly you can show output in a file instead of printing output on terminal
  - Personalizing the environment: You can define your own command

Filename Shorthands

Ch 1.1
Ch 1.2
Ch 1.3
Ch 2.1
Ch 2.2
Ch 2.3
……
……

- pr ch 1.1 ch 1.2 ch 1.3 ……

- pr ch *
- wc ch 1.*

- ls ? //list files with single character name
- rm ? //remove files with single character name

Input-Output Redirection

- ls > filelist

- cat f1 f2 f3 > temp
- cat f1 f2 f3 >> temp

- sort temp
- sort <temp
- sort
  asd
  fgh
  bre
  CTL+d
  asd
  bre
  fgh

Personalizing Environment

- stty erase e
- stty kill k

File System

- A file a sequence of bytes
- 'od' command prints a visible representation of all bytes of a file

- $ od –c junk //interpret bytes as characters

```
0000000 n o w  i s  t h e  t i m e  \n
0000020 f o r  a l l  g o o d  p e o
0000040 p l e /n
0000044
   $
```

File System

- $ od –cb junk //interpret bytes as characters and octal numbers as well

```
0000000 n o w  i s  t h e  t i m e  \n
        156 157 167

0000020 f o r  a l l  g o o d  p e o

0000040 p l e /n

0000044
   $
```

- Here every character associated with a octal number
- The associated octal number represents the ASCII value of the character

What is in a file?

- The file command determines the file type of a file
- It reports the file type in human readable format
- Example:

```
$ file file1.txt
file1.txt: ASCII text
$
```

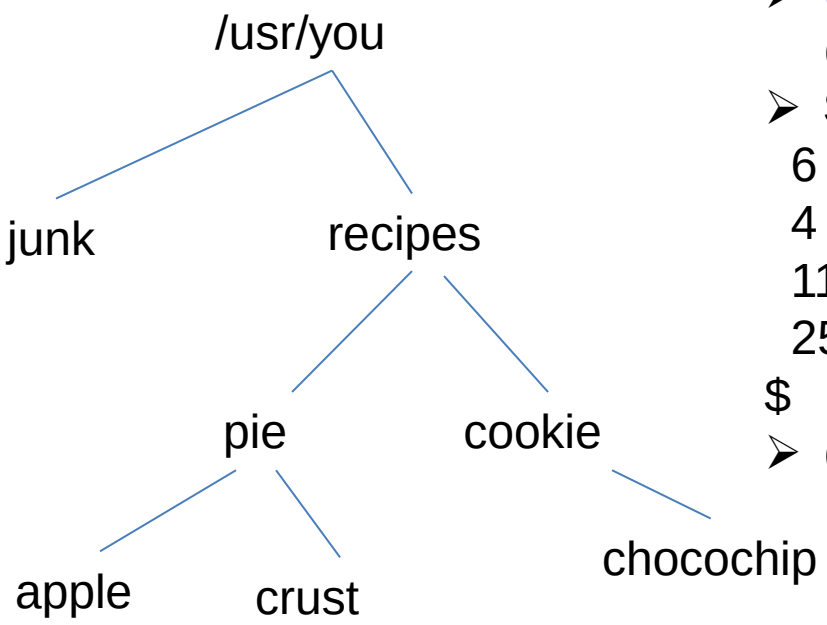
- To determine the types, file did not pay attention to the names
- File reads the first few hundred bytes of a file and looks for clues to the file type
- For example, file looks for words #include to identify C source

Directories and Filenames

- Files have unambiguous names, starting with path from root to current directory then the name you provided to the file
- Suppose, your current directory is ‘you’, then unambiguous name of any file of ‘usr’ is

  /usr/you/filename // here, /usr/you – is the path from ‘root’ to ‘you’
- But if you type ls, it does not print /usr/you/filename, the filename printed without any prefix
- Because, each running command or each process, has a current directory, and all filenames are implicitly assumed to start with the name of that directory
- So, it’s not show the path before filename explicitly

Directories and Filenames



- du command tell how much disk space is consumed by the files in a directory
- $ du

```
 6      ./recipes/pie          //Here numbers are the number of disk blocks
 4      ./recipes/cookie       // Typically each disk block has 512 0r 1024 bytes each
 11     ./recipes
 25     .
$
```

- du –a command print out all the files in a directory

Passwd File

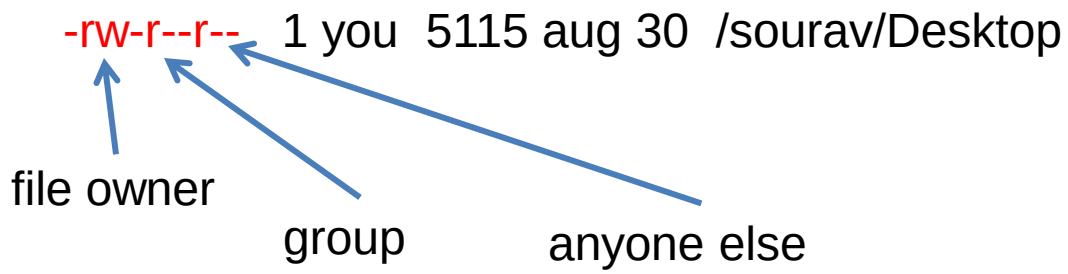
- The file /etc/passwd is the password file, it contain all the information about each user
- You can discover your uid and group-id by looking up your name in /etc/passwd

```
$ grep you /etc/passwd
you:gkmp987ghCOM:604:1:Y.o.a:/usr/you:
$
```

- login-id:encrypted-password:uid:group-id:miscellany:login-directory:shell
- The shell filed remain often empty, implying that you use the default shell

Permissions

- There are three kind of permissions associated for each file: read, write and execute
- Different permission associated with different people
- $ ls –l junk



Chmod

- chmod command changes permission on files
- Syntax: chmod options mode filename
- Options: -R(recursive), -f(force)
- Mode: read(r), write(w), execute(x)
- Two ways to specify mode for file:
  - Octal
  - Symbolic

Octal Mode

- **Owner:** r(4), w (2), x(1) **Group:** r(4), w (2), x(1) **Others:** r(4), w (2), x(1)
- Example: chmod -R 774 f1 //here 7 for owner, 7 for group and 4 for others
- r(4)+w(2)+x(1) = 7, means all the 3 modes are enable here
- Only 4 means, only read [r(4)] is enable
- Suppose, you put 6 for one member, mean r(4)+w(2) enable for that member

Symbolic Mode

- Syntax: chmod referencecs operator modes file
- References: **u**: owner, **g**: group, **o**: others, **a**: all
- Operator: **+** : add mode, **-** : delete mode
- Mode: **r** : read, **w** : write , **x** : execute
- Example : chmod u+rx filename // Enable read and execute mode for owner

Inodes (Index-Nodes)

- An Inode is a data structure that stores the following information about a file :
  - Size of file
  - Device ID
  - User ID of the file
  - Group ID of the file
  - The file mode information and access privileges for owner, group and others
  - File protection flags
  - The timestamps for file creation, modification etc
  - link counter to determine the number of hard links
  - Pointers to the blocks storing file’s contents

Inodes (Index-Nodes)

- ls –i command reports inode number of every files in a current directory
- $ ls –i

  15768 junk

  15274 recipes

  15852 x

  $
- df –ih command reports total no of inodes usages, free for every file systems
- stat filename: command reports inode information associated with the given file

Inodes Link

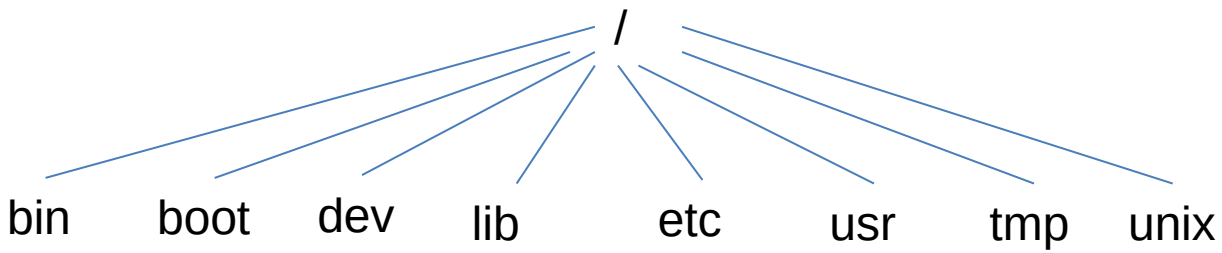
- More than one file can link to the same inode
- ln command makes a link of a new file with existing file:

  $ ln old_file new_file
- Now, new_file also point to the same inode of old file
- All same inode files has same permission mode
- If you change permission mode for one file, it will reflect to all files associated with that particular inode

Operation Using Inode

- Renaming a file does not change it's inode information
- Copied file has different inode number, its not same with original file
- Find a file using it's inode number : find . –inum inode_no
- find . –inum inode_no –delete // it will find and delete the file associated the particular inode

The Directory Hierarchy



- /unix is the program for the unix kernel itself
- /bin is the directory where the basic command programs reside
- /etc contains various administrative files like password file
- /lib contains c compiler and associated library
- /tmp is a directory for short-lived files created during the execution of the programs
- /usr contains user file system

Devices

- /dev contains peripheral devices files
- Peripheral devices : discs, tape, printer, keyboard, mouse
- Suppose, to read magnetic tape there is a file called dev/mt0 get executed to access the tape

grep Command

- grep pattern * -- it will give all the file name along with line which contain the pattern
- grep –l pattern * -- it will show only the file name which contain the pattern
- grep s*v junk – it will search for a pattern which start with s end with v in junk
- grep –v unix junk – it will print those line which not contain the word ‘unix’
- grep Unix junk – its case sensitive it will search for uppercase ‘U’
- grep –i Unix junk – its ignore case sensitive
- grep –c unix junk – it will print number of line count where unix appears

- If you want to search for a pattern with space-separated you must put within double-qoutes

grep Command

- grep “c[aeiou]ll” junk – it display all the start with ‘c’ and end with ‘ll’ and middle any character among a, e, i, o , u
- grep “^u” junk – it will search for line which start with ‘u’
- grep “^[aeiou]” junk– it will search for line which start with any character among a,e,i,o and u
- grep “^[^aeiou]” junk – it will work as reverse
- grep “u$” junk – it will print those line which end with ‘u’
- grep “^$” junk – it will print all the blank lines

Stream Editor: sed

- sed apply some instruction on selected line or selected pattern
- sed –n 1p junk – Print 1st line of junk
- sed –n 1,4p junk – print 1st to 4th line
- sed –n –e 1,4p –e 6,8p junk – print 1st to 4th line along with 6th to 8th line
- sed 2q junk – print first 2 line then quit, no other line print
- sed ‘s/t/T/g’ junk – substitute ‘t’ with ‘T’ globally, means in every position of the file
- sed ‘s/t/T/’ junk- substitute ‘t’ with ‘T’ only in the first position of ‘t’ in a line

❖ All this changes are temporary -- the changes are only show on terminal.
But if you want to change in original file you need to provide ‘- i’ in option

- Example:

sed –i ‘s/t/T/g’ junk

Stream Editor: sed

- sed ‘3 s/t/T/g’ junk – substitute ‘t’ with ‘T’ globally in 3rd line
- sed ‘1,3 s/t/T/g’ junk- substitute ‘t’ with ‘T’ from 1st line to 3rd line
- sed ‘2,$ s/t/T/g’ junk- substitute ‘t’ with ‘T’ from 2nd line to last line
- sed ‘/unix/ s/is/was/g’ junk- substitute ‘is’ with ‘was’, where the pattern unix is found
- sed ‘/unix/ a new_line’ junk – it will add new line after every line where pattern ‘unix’ found
- sed ‘/unix/ i new_line’ junk – it will add new line before every line where pattern ‘unix’ found
- sed ‘/unix/ c change the line’ junk – it will change the line where ‘unix’ pattern found
- sed 2d junk – it will delete the 2nd line

awk

- awk is a programmable, pattern matching and processing tool available in UNIX

- Alfred=a, weinberger= w , Kerinighan=K

- awk utility is a pattern scanning and processing language. It search one or more files to see if files contain specific lines that contain match pattern and then perform associated actions such as writing the to the standard output or incrementing a counter each time when it finds a match etc.

- Syntax : awk option ‘selection_criteria {action}’ filename
  Example: awk ‘/manager/ {print}’ emplist

** If no selection criteria is used, then action applies to all lines of the file

awk

- Selection Criteria: It is the filtering condition
- Actions: It will execute for all the selected pattern, action might be a print statement, assignment statement, arithmetic manipulation, loop, increment/decrement operator, awk built in function etc.

- $0 – entire line, $1 – first column, $2 – Second column and so on
- NR – line number in the given input file, NF – number of column of each line

- Arithmetic Operator : +, -, /, *, %
  Logical Operator: && , || , !
  Comparison Operator : >, <, ==, >=, <=
  Increment / Decrement Operator: ++, --

- You can use loops, condition within the action part

awk

- awk ‘/OS/ {print $1, $2, $3}’ junk
- awk ‘/[Bb]iswa[sl]/ {print $0}’ junk
- awk ‘NR==3, NR==6 {print NR, $0}’ junk-
- awk ‘$3==“OS” {print $1, $2, $4}’ junk
- awk ‘$3==“OS” || $3=“USP” {print $1, $2, $4}’ junk
- awk ‘$2~ ’[Bb]iswa[sl]’ {print $0}’ junk

- awk ‘$5>2000 {print $0}’ junk
- awk ‘$5>2000 {print $0, $5*.25}’ junk

awk

- Create your own ‘awk’ subprogram:

  Create a file with ‘awk’ extension. Example: sal.awk

  - Suppose, your sal.awk print name of those faculties have salary more than 5000 contain of sal.awk : $5>5000{print $1, $2}
  - Now run your ‘awk’ subprogram:

    awk –f sal.awk filename

awk

- awk ‘$5>5000 {count++, print count, $2, $3, $4, $5}’ junk
- awk ‘NF!=5 {print “line no”, NR, “has”, NF, “columns”}’ junk

awk

- BEGIN and END in awk:
  awk ‘BEGIN {actions} {processing statement} END {actions}’
- awk ‘BEGIN {count=0} {count++} END {print “total no of lines::” count} ’ faculty.txt
- Find the number of faculty teaching USP?
- Print the average salary of all faculty?
- awk built in variables : NR, NF, RS , FS, OFS, ORS, ARGC, ARGV

wait() and waitpid()

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait( int *statusPtr);

 pid_t wait(pid_t pid, int *statusPtr, int options);
```

- The wait() function causes a parent caller to suspend execution until a child's status becomes available
- The waitpid() suspends the calling process until a specified process terminates

wait() and waitpid()

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait( int *statusPtr);

 pid_t wait(pid_t pid, int *statusPtr, int options);
```

- If pid is greater than 0, the calling process waits for the process whose process identification number (PID) is equal to the value specified by pid.
- If pid is equal to 0, the calling process waits for the process whose process group ID is equal to the PID of the calling process.
- If pid is equal to -1, the calling process waits for any child process to terminate.
- If pid is less than -1, the calling process waits for the process whose process group ID is equal to the absolute value of pid.

wait() and waitpid()

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait( int *statusPtr);

 pid_t wait(pid_t pid, int *statusPtr, int options);
```

- Options specifies optional actions for the waitpid function. Either of the following option flags may be specified, or they can be combined with a bitwise inclusive OR operator:
- WNOHANG causes the call to waitpid to return status information immediately, without waiting for the specified process to terminate. Normally, a call to waitpid causes the calling process to be blocked until status information from the specified process is available; the WNOHANG option prevents the calling process from being blocked. If status information is not available, waitpid returns a 0.
- WUNTRACED causes the call to waitpid to return status information for a specified process that has either stopped or terminated. Normally, status information is returned only for terminated processes.

wait() and waitpid()

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait( int *statusPtr);

 pid_t wait(pid_t pid, int *statusPtr, int options);
```

- statusPtr is a pointer to the location where status information for the terminating process is to be stored. There is one situation when status information is not available after a call to waitpid: if waitpid is called with NULL as the statusPtr value, no status information is returned.

wait() and waitpid()

- After the call to waitpid, status information stored at the location pointed to by statusPtr can be evaluated with the following macros:
- WIFEXITED(*statusPtr): evaluates to a nonzero (true) value if the specified process terminated normally.
- WIFSIGNALED(*statusPtr): evaluates to a nonzero (true) value if the specified process terminated because of an unhandled or unknown signal.
- WIFSTIPPED(*statusPtr): evaluates to a nonzero (true) value if the specified process is currently stopped but not terminated by known signal.

exec()

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

int execl(const char *path, const char *arg,.../* (char *) NULL */);
int execlp(const char *file, const char *arg,.../* (char *) NULL */);
int execle(const char *path, const char *arg, .../*, (char *) NULL, char * const envp[] */);
int execv (const char *path, char *const argv[]);
int execvp (const char *file, char *const argv[]);
int execve (const char *path, char *const argv[], char *const envp[]);
```

- The exec family of functions replaces the current running process with a new process.
- It can be used to run a C program by using another C program.

open()

```
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>
int open (const char* Path, int flags , int mode );
```

- **Path :** path to file which you want to useIt can be used to run a C program by using another C program.
- **flags :**
  - **O_RDONLY**: read only, **O_WRONLY**: write only, **O_RDWR**: read and write,
    O_CREAT: create file if it doesn't exist, **O_EXCL**: prevent creation if it already exists

close()

```
#include <fcntl.h>
int close(int fd);
```

- ‘0’ on success and ‘-1’ on failure

read()

```
#include <fcntl.h>
size_t read (int fd, void* buf, size_t cnt);
```

‘**fd:** file descripter
buf: buffer to read data from
cnt: length of buffer

- return Number of bytes read on success

write()

```
#include <fcntl.h>
size_t write (int fd, void* buf, size_t cnt);
```

‘**fd:** file descripter
buf: buffer to write data from
cnt: length of buffer

- return Number of bytes write on success

select()

```
#include <sys/select.h>
#include <sys/time.h>

 int select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timeval *timeout);

/* Returns: positive count of ready descriptors, 0 on timeout, –1 on error */
```

- struct timeval

```
{
 long tv_sec; /* seconds */
 long tv_usec; /* microseconds */
};
```

select()

```
void   FD_ZERO (fd_set *fdset);        /* clear all bits in fdset */
void   FD_SET (int fd, fd_set *fdset);   /* turn on the bit for fd in fdset */
void   FD_CLR (int fd, fd_set *fdset);    /* turn off the bit for fd in fdset */
Int    FD_ISSET (int fd, fd_set *fdset);   /* is the bit for fd on in fdset ? */
```

```
fd_set  rset;
FD_ZERO(&rset);   /* initialize the set: all bits off */
FD_SET(1, &rset);  /* turn on bit in rset for fd 1 */
FD_SET(4, &rset);  /* turn on bit in rset for fd 4 */
FD_SET(5, &rset);  /* turn on bit in rset  for fd 5 */
```