System Programming Notes

Chapter # 6: Unix Special Files.

→ Interprocess communication allow process running on the same system to share information and hence co-operate.

→ Pipes and Fifo's are example of special files used in interprocesses communications.

→ Synopsis:

```
#include <unistd.h>
int pipe (int fildes[2]);
```

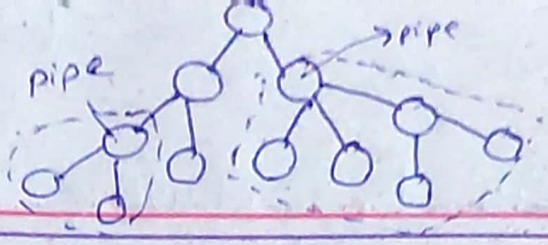
This function creates a communication buffer that caller can access through the file descriptors.

~ fildes[2]: file descriptors. Data is written to fildes[1] and can be readed from filedes[0], on a first-in-firstout basis

*s 0

*us -1 and set errno.

(2)



→ Pipe has no external or permanent name, so a program can access it only through its two file descriptors.

→ A pipe can only be used by the process that created it and by descendants that ~~excd~~ inherit the descriptors on fork.

→ Usually a parent process uses pipes to communicate with its children.

→ Pipe file is destroyed when no process have opened it.

→ From pipes data is erased when another process read it.

→ When a writ to pipe is done and it is now the turn to read. We should take care that the process shouldn't read it written

content again. For this we have two solutions:

- sleep for some time when we have to read after write.
- Take two pipes one for reading and one for writing.

6.3 FIFO's

→ Pipes are temporary, but fifo's or named pipes by special files that persists even after all process have closed them.

→ A FIFO has a name, permissions and appear in directory listing given by 'ls' just like ordinary files.

→ Synopsis:

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

This function creates a new fifo special file.

~path : the path of fifo file
~mode : permissions for newly created fifo.

*S 0
*US -1 and set errno.

→ We can remove fifo the same way as an ordinary file, either by 'rm' or 'unlink'.

→ Use for communication b/w any two independent processes.

6.4 Pipes and the Client-Server Model:

→ Standard pattren for process interaction

→ One process, designated as the client, request a service from another process, called the server.

→ A malicious client could cause the protocal to behave incorrectly without detecting an error.

→ In most cases, the client should never be able to cause the server to fail or exit.

→ When a client/server first ~~write to~~ a fifo/pipe and then it has to read from pipe/fifo in most cases it reads its own content. To solve this we have two solution:

- Apply a sleep for some time after writing and before reading
- Create two seperate fifo/pipes to read and writ.

→ Sometime a partial read by one client causes the next ~~to~~ client to receive

(6)

incorrect results, or the problem discuss above of malicios client. To salve these two problems, a ~~ete~~ server should create a seperate named pipe for each distinct ~~f~~ client.

→ 'stty -a' command disply the current terminal settings.