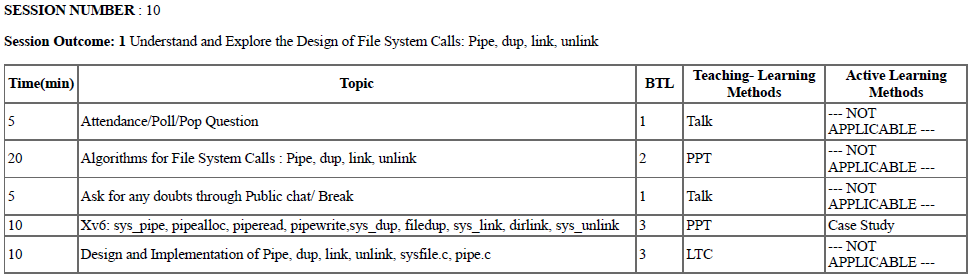
**Operating Systems Design  
19CS2106R​  
Session-10**  
**Lecture Notes  
File System Calls**



**Learning Outcomes**

At the end of the session, the student will

* ***Understand the system calls pipe ( ), dup ( ), link ( ) and unlink ( ).***
* ***Understand the xv6 case study of the above system calls.***
* ***To explore the design and implementation of the above in sysfile.c***

**Concept of Pipes**

* Pipes are the IPC mechanism used for communicating between two or more process in Linux i.e to pipe the data between TWO processes.
* It is a form of redirection (transfer of standard output to some other destination).
* Their implementation allows processes to communicate even though they do not know what processes are on the other end of the pipe.
* Two types of pipes:

**Unnamed pipes** and **Named pipes**

* Pipe is used to combine two or more commands, and in this, the output of one command acts as input to another command, and this command’s output may act as input to the next command and so on.
* Piping mechanism is denoted by the ‘|’character.
* The command line programs that do the further processing are referred to as ***filters***.

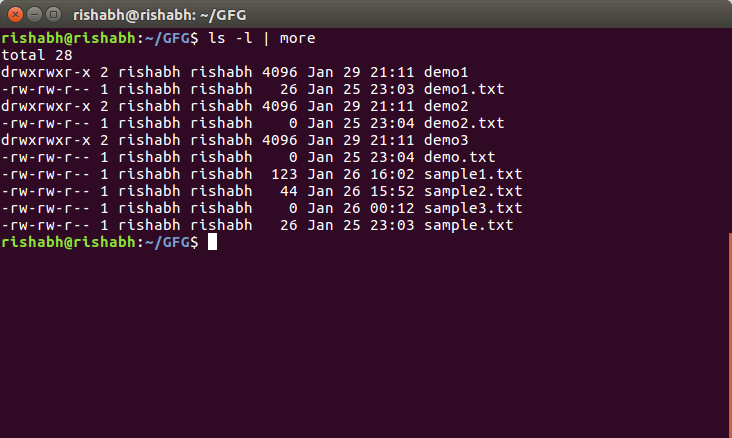
**General Syntax:**

* Command\_1 | command\_2 | command\_3 | .... | command\_N
* Eg : **Listing all files and directories and give it as input to more command.**

**$** ls -l | more .

Here two commands namely, ***ls-l*** and more are piped. ***ls*** command lists and ***more*** command displays the content one page at a time.

The ***more*** command takes the output of ***$ ls -l*** as its input. The net effect of this command is that the output of ls -l is displayed one screen at a time. The pipe acts as a container which takes the output of ls -l and gives it to more as input.



**Unnamed Pipes**

1. These are created by the shell automatically.  
   2) They exists in the kernel.  
   3) They can not be accesses by any process, including the  
   process that creates it.  
   4) They are opened at the time of creation only.  
   5) They are unidirectional.

**Named Pipes( also called FIFO, First In FIrst Out)**

* They have “names” and exist as special files within a file system.
* They are created programmatically using the command ***mkfifo*** or will be created using ***mknod()*** system call..
* They exist until they are removed with ***rm*** or ***unlink() .***
* ***Normally used to communicate between TWO totally unrelated processes.***
* ***These are bi-directional.***

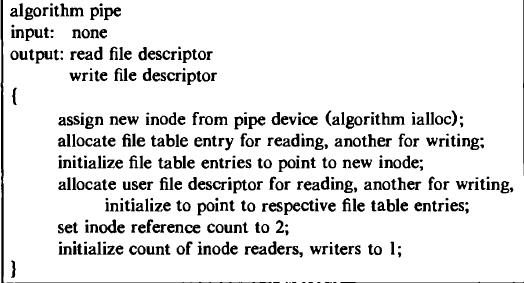
**Syntax of *pipe ( )* system call**

* The syntax for creation of a pipe is

***pipe (fdptr) ;***

where ***fdptr*** is the pointer to an integer array that will contain the two file descriptors for reading and writing the pipe.

**Algorithmic Design Approach of creation of *pipe( )* system call-unnamed**



***dup ( )* System call**

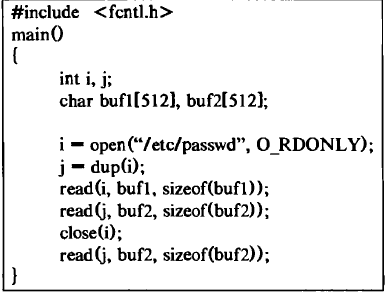
* The dup system call copies a file descriptor into the first free slot of the user file descriptor table, returning the new file descriptor to the user.
* The syntax of the system call is

***newfd -= dup (fd);***

where fd is the file descriptor being duped and newfd is the new file descriptor that references the file.

Because dup duplicates the file descriptor, it increments the count of the corresponding file table entry, which now has one more file descriptor entry that points to it.

**Design of *dup ( )* System call-C Procedure**



**Creating new link to an existing file using *link ( )* system call**

* The link system call links a file to a new name in the file system directory structure, creating a new directory entry for an existing mode.
* The syntax for the link system call is

***link(source file name, target file name);***

where source file name is the name of an existing file and target .file name is the new (additional) name the file will have after completion of the link call

**Algorithmic Design for link () system call**

* **Algorithm link**
* **input: existing file name**
* **new file name**
* **output: none**
* **{**
* **get inode for existing file name (algorithm namei);**
* **if (too many links on file or linking directory without super user permission)**
* **{**
* **release inode (algorithm iput);**
* **return (error);**
* **}**
* **increment link count on inode;**
* **update disk copy of inode;**
* **unlock inode;**
* **get parent inode for directory to contain new file name (algorithm namei);**
* **if (new file name already exists or existing file, new file on**
* **different file systems)**
* {
* **undo update done above;**
* **return (error); }**
* **Create new directory entry in parent directory of new file name:**

**include new file name, inode number of existing file name;**

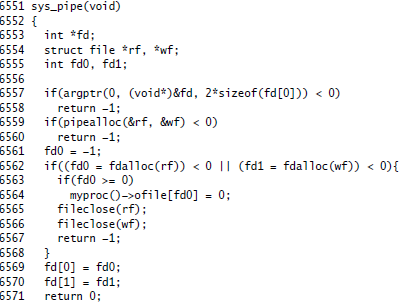
* **Release parent directory inode (algorithm iput);**

**release inode of existing file (algorithm iput);**

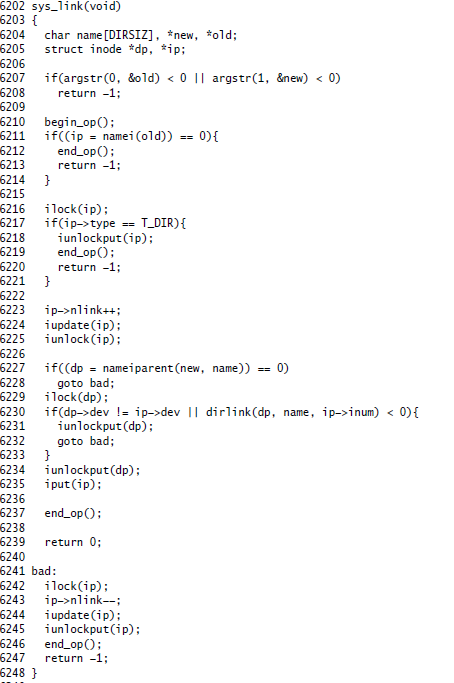
}

* Xv6 Case Study : ***sys\_pipe, pipealloc, piperead, pipewrite, sys\_dup, filedup, sys\_link and sys\_unlink.***

**Function of pipe ( ) system call using xv6**



**Function of link ( ) using xv6**



**Removing a directory entry for a file using Unlink ( ) system call**

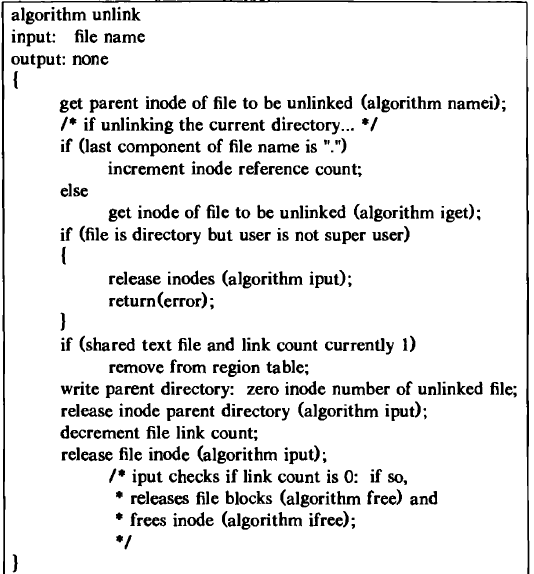
* The unlink system call removes a directory entry for a file.
* The syntax for the unlink call is

***unlink (pathname) ;***

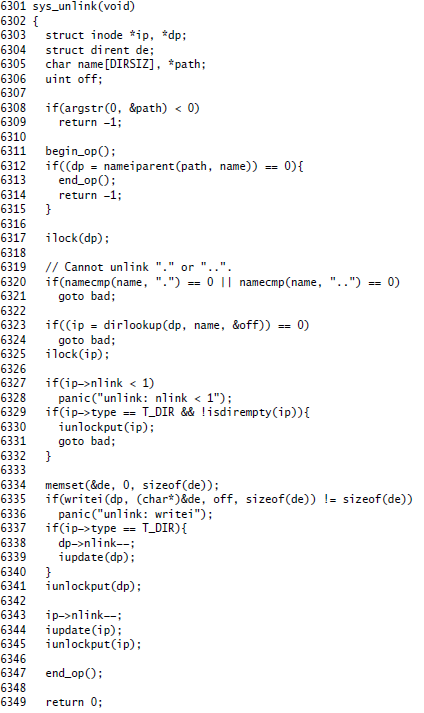
where pathname identifies the name of the file to be unlinked from the directory hierarchy.

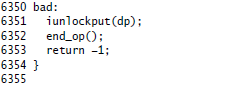
* If a process unlinks a given file, no file is accessible by that name until another directory entry with that name is created.

**Algorithmic Design of *unlink ( )***

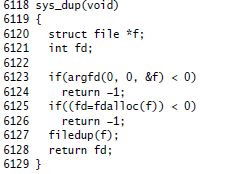


**Function of of *unlink( )* in xv6**





**Function of *dup ( )* using system call using xv6**



* Design and Implementation of ***sysfile.c***

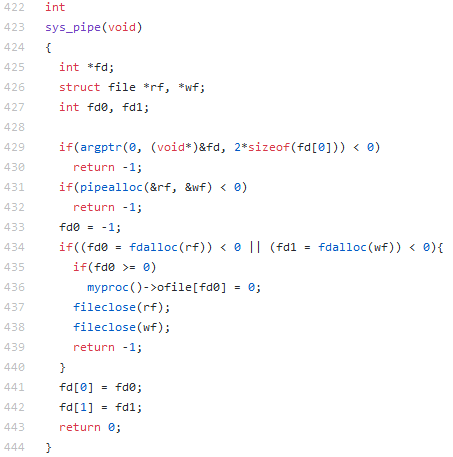
***https://github.com/mit-pdos/xv6-public/blob/master/sysfile.c***

***sysfile.c*** is a github repository on various file system calls.

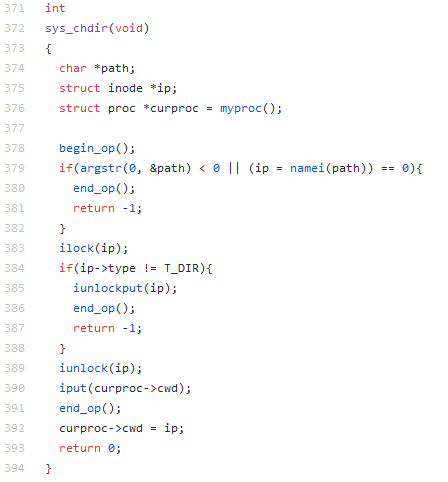
* Consists of 444 lines of xv6 code.

***sysfile.c* file consists of xv6 implementations of**

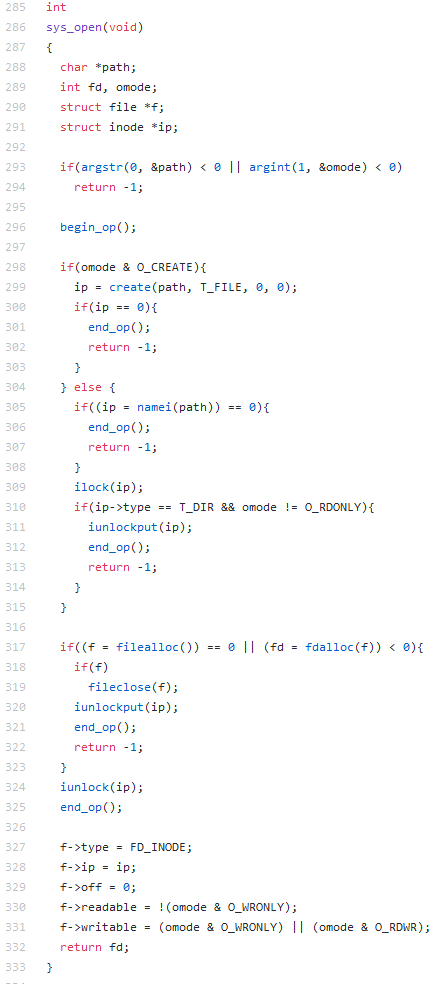
* ***fdalloc( )---for file descriptor allocation***
* ***Sys\_dup---dup system call***
* ***Sys\_open—open system call***
* ***Sys\_read---read system call***
* ***Sys\_write---write system call***
* ***Sys\_close---close system call***
* ***Sys\_fstat----fstat system call***
* ***Sys\_link----link system call***
* ***Sys\_unlink---unlink system call***
* ***Sys\_mknod---mknod system call***
* ***Sys\_chdir---Chdir System call***
* ***Sys\_pipe-----Pipe System Call***



**sys\_chdir**



**Sys\_open**

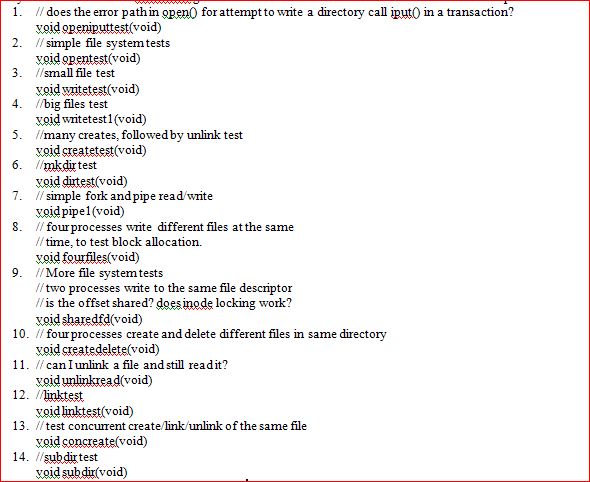


* **Conduction of ALMs in Breakout Session**

**ALM-1**

* ***Understand the internal algorithms behind the design of various xv6 file system calls.***
* ***Perform the given tests related to file system.***
* ***Customize the usertests.c given in xv6 source code base and execute.***
* ***Submit the output of all the tests***

**ALM-1**



**ALM-2**

* Assume that a process A executes the following three function calls:

fdl = open(“/etc/passwd”, O\_RDONLY);

fd2 = open(“local”, O\_RDWR);

fd3 = open(“/etc/passwd”, O\_WRONLY);

For process A, show the relationship between the inode table, file table, and user file descriptor data structures.

Suppose a second process B executes the following code.

fdl — open("/etc/passwd", 0\_RDONLY);

fd2 — open("private", 0\_RDONLY);

***Draw the resulting picture that shows the relationship between the appropriate data structures while both processes (and no others) have the files open.***

**ALM-3**

* The following sequence of code has been observed in various programs:

***dup2(fd, 0);***

***dup2(fd, 1);***

***dup2(fd, 2);***

***if (fd > 2)***

***close(fd);***

***To see why the if test is needed, assume that fd is 1 and draw a picture of what happens to the three descriptor entries and the corresponding file table entry with each call to dup2. Then assume that fd is 3 and draw the same picture.***

**ALM-4**

* ***Write a program that create two pipes, send filename from command line to child process. In child read that file and send it back using pipe. Parent process should print the file. if error occur in child process error must be send to parent process***