**Operating Systems Design 19CS2106S**

**Session – 9 &10**

**ALM**

1. Understand the internal algorithmsbehind 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
2. // does the error path in open() for attempt to write a directory call iput() in a transaction?

voidopeniputtest(void)

1. // simple file system tests

voidopentest(void)

1. //small file test

voidwritetest(void)

1. //big files test

voidwritetest1(void)

1. //many creates, followed by unlink test

voidcreatetest(void)

1. //mkdir test

void dirtest(void)

1. // simple fork and pipe read/write

voidpipe1(void)

1. // four processes write different files at the same

// time, to test block allocation.

voidfourfiles(void)

1. // More file system tests

// two processes write to the same file descriptor

// is the offset shared? does inode locking work?

voidsharedfd(void)

1. // four processes create and delete different files in same directory

voidcreatedelete(void)

1. // can I unlink a file and still read it?

voidunlinkread(void)

1. //linktest

voidlinktest(void)

1. // test concurrent create/link/unlink of the same file

voidconcreate(void)

1. //subdir test

voidsubdir(void)

1. // test writes that are larger than the log.

voidbigwrite(void)

1. //bigfile test

voidbigfile(void)

1. // DIRSIZ is 14.fourteen test

voidfourteen(void)

1. //rmdot test

voidrmdot(void)

1. // dir vs file test

voiddirfile(void)

1. 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 filedescriptor data structures.

Suppose a secondprocess 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 whileboth processes (and no others) have the files open.

1. 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 fdis 1 and draw a picture of what happens tothe three descriptor entries and the corresponding file table entry with each call to dup2.Then assume that fdis 3 and draw the same picture.

1. 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.