P536: ADVANCED OPERATING SYSTEMS

FILE SYSTEM ASSIGNMENT

TEAM: ROHIT PATIL, SAMEEKSHA VAITY

Files created / modified for this assignment :

1. shell -> xsh_fstest.c
2. include -> fs.h
3. system -> fs.c , bs.c
4. shprototypes.h -> to include the command fstest
5. shell.c -> to include the command fstest

Implementation Description:

The file fs.c contains the following functions which form the basis of in memory file systems implementation :

1. int fs_create(char *filename, int fopen_mode)
   This function first checks if the file is open in the create mode then it checks if it already exists in the directory. If it does not find a file with the same name then it searches for a free node and creates a new entry in the open file table (OFT), mark the file as open, and create a new dirent and inode that represent this new file. These changes are reflected back to the OFT.

2. int fs_open(char *filename, int flags)
   Given a filename, fs_open validates the inode information , allocates the file with the file open file descriptor , updates the file table , appends flags to the file table , and updates the inode information.

3. int fs_close(int fd)
   Given a file descriptor fd, fs_close simply looks at the OFT and marks the specified file as closed. It updates the inode information and saves it to the memory.

4. int fs_seek(int fd, int offset)
   Given a file descriptor fd and offset, fs_seek will set the pointer in the file according to the specified offset. First it gets the new file pointer location then makes a check to see if it goes beyond the size then updates the old file pointer location to the new file pointer location.

5. int fs_read(int fd, void *buf, int nbytes)
   Given a file descriptor fd, buffer buf, and nbytes, reads nbytes bytes of data from the file specified by fd into buf. Fisrt it checks if the size of bytes to be read with the file size if within range it proceeds. This is done by first calculating the inode block by dividing the filepointer by the block size.Then it finds the offset within a block by performing (filepointer mod block size). If nbytes is not greater than the difference between the block size and the filepointer offset, fread

will simply read nbytes from the offset. Otherwise, when fread reaches the end of the block, it will read the rest of the nbytes from the next block.

6. int fs_write(int fd, void *buf, int nbytes)
   fs_write does the same thing as fs_read, of writing the bytes of the file specified instead of reading them.

Below is the output after running the fstest command in XINU shell;

```
Output
=======
xsh $ fstest -f
1100000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000


 Obtained inode.

 Size updated to 3 blocks.
 1500 bytes written!

 Data read:
 tuvwxyz{|}~!"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+
,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+
,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+
,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+
,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
 Data read 477 bytes.
1100000000000000001110000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
```

Lessons Learned:
How to implement In memory file system in XINU and obtained a firm understanding of implementation details of the Unix filesystem structure.

Team Contributions:

We both have equally contributed for the the completion of this assignment.