Design Document
Implementation of File System on XINU OS:

This project is an implementation of a File System written in C for operating system XINU. The code was run successfully with XINU OS deployed on RPI node 21.

Introduction:

File system offers a series of interfaces to perform operations on the disk of a system. These interfaces include system calls to create a file, open a file, write to a file, read from a file, append to a file, close a file, delete files etc.

Implementation of the File System:

Structures:

a) **File Table : *struct filetable***
   This table contains entries for each opened file. It has an attribute called state which identifies whether a file is opened for read(0) or write(1) or append(2). It has entries to the hard links of the file as well as the inode entry in the Inode table for the file.

b) **Directory Table: *struct dirent***
   This table contains entries for each file present within a directory. It has "filename", "inode number" pairs, which are used to map filenames to inodes. It contains one entry per file. inode_number -1 signifies that the entry is free.

c) **Inode Table:**
   This table contains entries for each Inode. The value in "type" suggests whether it is free or not.

**note : the above mentioned tables are created as array objects of structs*

Funtionality:

1. **Initializing the File System *mkfs()*:**
   This function initializes the struct fSystem which identifies various parameters of the file system like total no. of blocks, total no. of inodes, the size of a block etc. This struct also points to the root_directory. The program also initializes entries in file table, directory table and inode table.

2. **Creating a file(): *fCreate()***
   To create a file, we first create a new index in the Inode table by searching for a free entry(type = -1). This inode number is stored in the directory table dirent by storing "filename" and the inode index. So, we map the filename to its inode number here.

3. **Opening the file(): *fOpen()***
   While opening the file, we will search for a free entry in the file table and allocate it to the file. This is done by setting its type to "read" or "write" depending on the value received in the

"flags" parameter. The file can either be opened in readOnly or writeOnly mode. Either way, it has to be closed before it can be opened again.

4. **Reading from the file():*int fread(int fd, void *buf, int nbytes)***
   This function reads nbytes from the buffer *buf for the file descriptor fd. The file descriptor is the entry inside the table filetable. From fd we first check if the file is open in readOnly mode. If yes, then we fetch the single-direct blocks from the Inode entry in the Inode table. From these blocks, nbytes of data is read into the buffer and returned.

5. **Write into the file():*int fwrite(int fd, void *buf, int nbytes)***
   This function writes nbytes into the buffer *buf for the file descriptor fd. The entry from the file table is used to check if the file is open in writeOnly mode. If yes, then we fetch the entry of its inode in the inode table. The blocks attribute contains an array of all the blocks which need to be allocated to the file. The Bitmap is checked to identify the blocks which are free and these blocks are one by one stored in the array of blocks in the inode entry. These blocks are then written with the values from the buffer. If the file is overwritten and if the old blocks are to be freed then the bitmap is updated to 0 so that the blocks are free.

6. **fseek(int fd, int offset);:**
   fseek allows us to set the file pointer to a specified position. The file descriptor entry is accessed with 'fd'. This entry is used to update the fptr value with the offset needed to be set in the file. Subsequent read or writes would work from this value of the fptr. Fptr cannot be set beyond the last character of the file.

All these system calls were successfully tested using a test function written in xinu and run on RPI node 21.

Future enhancements:
The file system has only the basic commands. It can be extended to incorporate more interfaces for the file system like append and handling for directory within a directory. Also changes could be made to store inode table and filetable on the disk(heap in the current case which simulates the disk).