File Systems

A brief description of implemented code

File create:

- The create method takes up the globally initialized filetable instance and checks for the next available slot, if all the blocks are filled return error.
- Assign the next available slot to file descriptor and start initializing the structures dirent and inode with all the available values.
- I maintain separate variables for inode number and for the first data block to be used. Inode number starts with '0' and first data block will be '17'.
- Update the directory structure with number of entries and dirent structure array.

File open:

- Search for the file descriptor passed as an argument in the globally initialized array of filetable structures
- If found update the state and the flag, I have put an extra variable in filetable structure to maintain the mode with which file was opened.
- Return the file descriptor.

File write:

Now was the real test, I'll try to explain in brief

- First validate the flag, if the file was opened in read only mode, return error
- Take the file descriptor and access the respective inode
- Find out how many blocks of data the inode contain
- Find out the next free block through getmaskbit function call
- Handle the writing in such a way that the data is written without segmentation, if for example the file has already been written, continue writing from the same block. Also calculate how many blocks are needed to write the buffer
- The code works flawlessly to keep segmentation at bay.
- Finally set mask bits to update the data blocks used
- Return the number of bytes written

File close:

Change the state in file table by accessing the filetable structure through file descriptor passed.

system call. Hence the code to print the actual memory used by a process when compared to max memory allocated should be written in Kill system call.

As stated earlier, we have made arrangements to show this behavior using a special command in XINU.

File Read

For Reading a File, We used the File descriptor to retrieve the INODE structure of the file and used the same to calculate the number of blocks used by the file. Then, by using the same INODE we retrieved the data from the memory block and then copied the same to the buffer stream. Then, move on to the next block and retrieve the same. Once the data is fully read, it is copied into the buffer stream and then displayed using the same buffer.

File Seek:

Once the whole file is read, the file pointer will be present at the End of the File. So, in order to read the file again we have to move the file pointer to the starting of the file to read it correctly. This can be done by moving the file pointer through the size of the current blocks offset places. This is done by adding the file pointer value and negative of the offset value.

Mount:

Even though the Mount function cannot be used in our implementation, the mount function can be used to load the file system parameters if we were to use the file system implementation in primary- secondary memory structure. It would load all the file system parameters from the secondary memory to the primary memory.

Lessons learnt

The main learning curve for this assignment was the structure of file systems in an Operating system and the steps required from starting of file creation, file opening, file write and closing a file. Each function requires an implementation of certain structure to store all the metadata of the files and using the same to modify the files based on its parameters.

Contributions

Anuj – create, open, write, close, mount.

Took help from Prashant Balasubramani for seek and read functions