FILE SYSTEM IMPLEMENTATION IN XINU

Introduction:

The purpose of this document is to outline the technical design of the File system implementation in XINU operating system.

Design Considerations:

- Establish a basic file system with memory based backing restore.
- The implementation is a byte-stream file interface and we provide the following file and directory functions to maintain the abstraction:

  - **fopen:** Opens an existing file and brings it into memory so that the processes can use it. When a process calls open, it should locate the file the process identifies and return a file descriptor, which is the process's means of identifying a file.
  - **fclose:** Closes or release a file descriptor from use and make the file inaccessible until another open call is made.
  - **fread:** Read n bytes out of a file and into buffer space in a process's memory. When a read call is made, it should start where the previous read left off. To do this, we keep a seek pointer associated with the file.
  - **fwrite**: Write should let us write n bytes from a process's buffer memory into a file. If a write call would go past the end of a file, the file should be made larger to accommodate the new data. The write call should also use the seek pointer. New data written into a file should start at the seek pointer, and when write is complete the seek pointer should be updated.
  - **fseek:** Allows to move the seek pointer in a file, allowing us to choose which part of the stream we read and write to.
  - **fcreat:** It is used to create a new file.

Implementation details:

- int fopen (char *filename, int flags);

  fopen: It finds the inode corresponding to the filename and since the file is open it is, loaded into memory into the inode table. A new open file table entry has been created to refer to this file. The calling process has allocated a file table entry to point to the open file table entry. The index of the file table entry is returned to the process.

- int fclose(int fd);

  fclose: It uses the file descriptor from the system call and the file table to find the open file table entry for this file. When we close the file, the inode is removed from the table and written back to disk.

- int fcreat(char *filename, int mode);

  fcreat: It allocates a free inode to the file and any directories that need updating are dealt with in addition a new open file table entry is created to refer to this file. The index of the file table entry is returned to the process.

- int fseek(int fd, int offset);

  fseek: It finds the open file table and updates the seek pointer to the value specified.

- int fread(int fd, void *buf, int nbytes);

  fread: It uses the open file table to find the inode referred to. It then reads n bytes starting from the seek pointer into the process buffer memory, and updates the seek pointer. If the seek pointer is at the end of the file, EOF is returned to the process.

- int fwrite(int fd, void *buf, int nbytes);

  fwrite: It works similar to read and if write proceeds past the end of the file, additional blocks are found in a list of free blocks and added to the inode address entries.

  Inode structure:

```
struct inode
{
        int id;
        short int type;
        short int nlink;
        int device;
        int size;
        int blocks[INODEBLOCKS];
};
```

File table structure:

```
struct filetable
{
    int state;
    int fileptr;
    struct dirent *de;
    struct inode in;
};
```

Directory table structure:

```
struct directory {
int numentries;
struct dirent entry[DIRECTORY_SIZE];
};
```

Future enhancements:

- Implementing with directory hierarchy.
- Addressing Implementation using double-indirect and triple-indirect addressing.

Output

$xsh:fstest

Input would be welcome to the world of Xinu ,fread would read that and writes it into a buffer and then we print it.

References:

1.Text book Xinu – Operating system Design
2.http://www.cs.ucsb.edu/cs170/notes
3.Wikipedia