OS Lab Assignment 4

Mesharya M Choudhary
190101053

Video Link:-

https://www.youtube.com/watch?v=8fk3P_iEDwA

Features:-

(i)Extending the inode structure to include double indirect block and triple indirect blocks as well:-

Relevant files which will help us understand the structure of inode in xv6 and then implement additional features:-

- file.h- This file contains the definition of the struct for the in-memory copy of the inode.
- fs.h- This file contains the definition of the struct for the on-disk inode. Apart from that it also has the macros for block size(BSIZE), number of blocks pointed by direct blocks(NDIRECT), number of blocks pointed to by singly-indirect blocks(NINDIRECT) and the file size in terms of number of blocks(MAXFILE).
- fs.c- This file contains definitions of functions related to the inode.The function of our concern is the bmap function which when given the arguments pointer to an inode and an integer n.It returns the address of the disk block that is pointed to by the nth block(0-based indexing) in the given inode.If the integer n is lesser than NDIRECT then this means that our nth disk block is being pointed to by a direct block and we directly access its address using the addrs member of the inode struct.Otherwise it means that our nth disk block is being pointed to by a singly indirect block so first we need to load the singly indirect block into the

memory after which we index into it appropriately and return the address of the desired disk block.

By default the xv6 inode contains 12 direct blocks and one singly indirect block.The struct for inode contains a member which is an array named addrs(it points to addresses of disk blocks) whose size is NDIRECT+1 where NDIRECT is a macro for number of direct blocks which is 12 as defined in the fs.h file.The fs.h file contains the MAXFILE macro which basically is the largest possible file that can currently be stored in xv6.Its size is 12+128=140 disk blocks(128 from single indirect block as it can point to BSIZE/sizeof(uint) number of files where BSIZE is the block size 512).

Changes proposed:-

fs.h-Modify NDIRECT in fs.h and set it to 10 i.e. we sacrifice two direct blocks to make space for 1 doubly indirect and 1 triply indirect blocks

We change the MAXFILE macro in fs.h accordingly (10+128+128*128+128*128*128). Also we must change the size of the file system by changing the macro FSSIZE in param.h so that it is big enough to accommodate the number of disk blocks in the largest file possible.

fs.c-Next we proceed to modify the bmap function which is responsible for allocating the nth disk block to the inode and returning the address of the allocated block.Note:Disk block allocation is done by invoking the balloc function.

- **Case-1**

The value of n<10 then we allocate the disk block to the direct block

- **Case-2**
  The value of n<10+128 and above cases not true then we allocate the disk block to the singly indirect block
- **Case-3**
  The value of n<10+128+128*128 and above cases not true then we allocate the disk block to the doubly indirect block
- **Case-4**
  The value of n<10+128+128*128+128*128*128 and above cases not true then we allocate the disk block to the triply indirect block

Case-1 and Case-2 have already been dealt with in the default implementation of bmap.For case 3 (we perform bn-=NINDIRECT^2 to get the proper indexing) we read a disk block into the memory which is responsible for pointing to multiple singly indirect blocks.We index into the correct singly indirect block using value bn/NINDIRECT and once we are in the correct singly indirect block we index into the correct direct block using value bn%NINDIRECT and then read a disk block and return its address.For case 4 we perform bn-=NINDIRECT^3 to get proper indexing and then we read a disk block into the memory which points to multiple doubly indirect blocks and then we index into the appropriate doubly indirect blocks using the value bn/NINDIRECT^2 and then we set bn=bn%NINDIRECT^2 after which we proceed in a similar manner as case 3 to get to the final disk block and return its address.

(ii)Adding symbolic links to xv6.A symbolic link is a type of file that points to another file.They are also called “soft links”.

Relevant files which will help us implement this feature:-

- usys.S,user.h,syscall.h,syscall.c,sysfile.c - We would be adding a system call sys_symlink so we will modify the corresponding files
  - stat.h-We add a new file type T_SYMLINK which represents symbolic links.
  - sysfile.c-We would be modifying the sys_open function in so that it works with the symbolic link.We also modify the create function.

Changes proposed:-

- sysfile.c - We are supposed to have the target and path as the arguments for the symlink where the target is the filename which is to be pointed to and path is the filename for the symlink file.We need to store the target somewhere so we create a symlink file inode with the given path argument and use its data block for storing the target in its data block.We implement all of this by creating sys_symlink function in sysfile.c which is a system call.Corresponding to this system call we perform the necessary changes in the other files related to the creation of system calls.

  Apart from this we also have to modify the create function which is responsible for creation of the inode corresponding to a new file.Whenever we create a inode for a file we check whether a file with same name and type exists already, if so then we return its inode rather than creating a new one.

  We have to modify the sys_open function for the case when we are opening a symlink file.In this case it may so happen that the symlink file points to another symlink file and so on and finally ends at a proper file.In this case we must trace and reach the inode of the proper file.For this we read the data block where we stored the target for the inode and find the inode corresponding to the target if the target file is not a symlink we are done otherwise we continue doing so till we reach a file which is not a symlink file,from this point onwards we can continue with the operations that were originally being performed in this function.