Lab Assignment 4: Print Metadata of a File in xv6 System

Task:Write user-level program for xv6 that creates a file of a given size and prints disk block numbers and i-node number and its contents of the file created

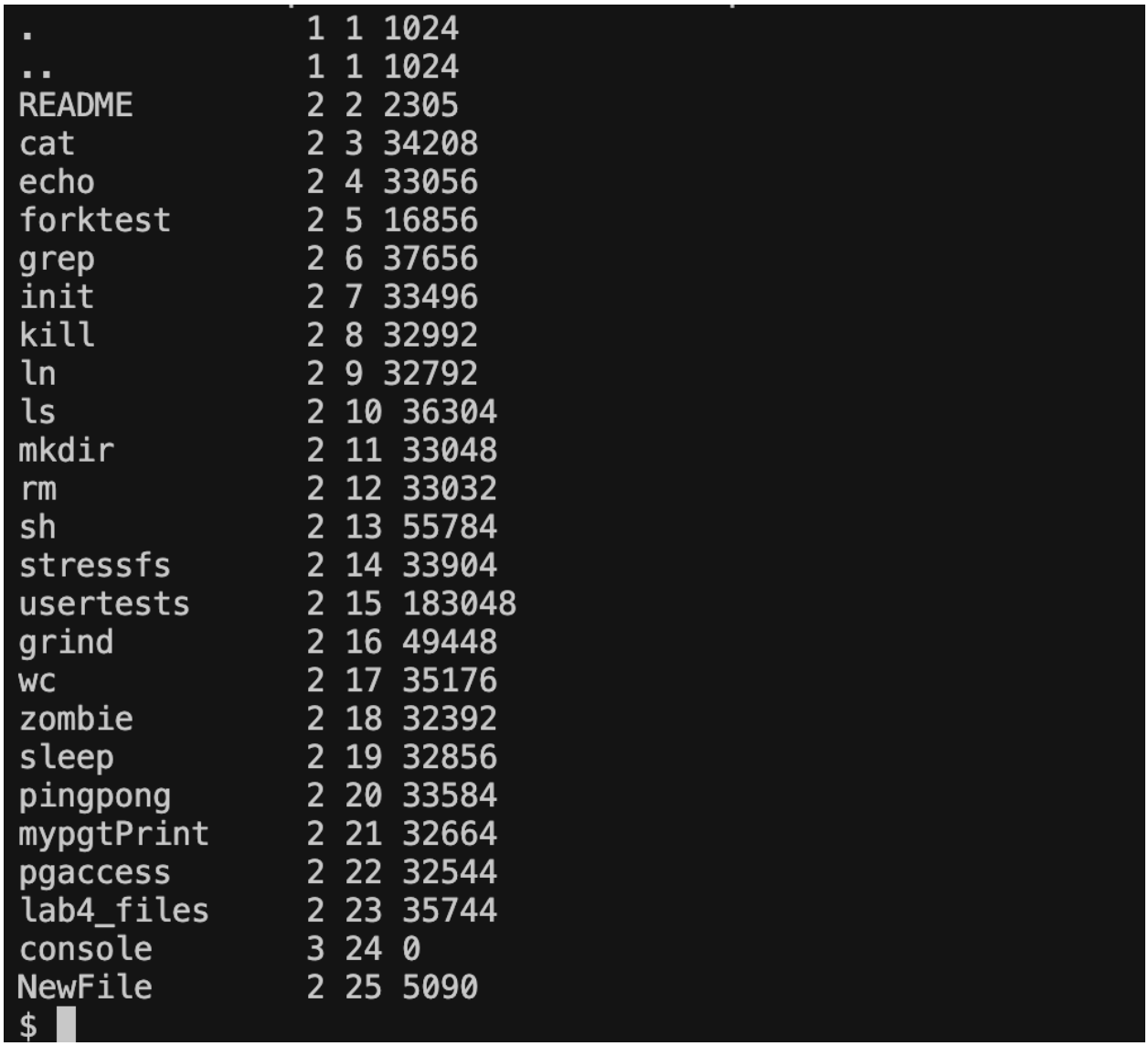
Methodology:

I have written a user program in user folder with name lab4_files.c.
In this program take the filename and filesize as command line arguments.
Using open() to open a file with given filename with O_CREATE|O_WRONLY flags.
If the file is already present then it will be opened otherwise new file will be created with access to write.
The current size of the file can be found out using size from structure stat.
If this current size is less than the given filesize then
a while loop is run till the current size becomes equal to given filesize
In this loop write() is called to write the roll number into every block.
In every iteration the size of the file is incremented by Block Size.
And last iteration the number of bytes written is adjusted so that the file is of given size.
After this write the file is closed and opened again this time with read and write permissions.
A structure stat st2 is defined and fstat() is called on file descriptor and this structure then st2 will contain information about the file.
st2.ino gives the inode number of the file this is printed.
The other attributes in the structure st2 are also printed.
A loop is run to print the block number and contents of each block.
This can be done by using a character array of BSIZE length, allocating memory to it using memset. read() is called to reach each block into this temporary buffer and this is printed.
Add this program to makefile as $U/_lab4_files\

OUTPUT:

```
$ lab4_files NewFile 5090
pagetable 0x0000000087f51000
.. 0: pte 0x0000000021fd5401 pa 0x0000000087f55000
.. .. 0: pte 0x0000000021fdbc01 pa 0x0000000087f6f000
.. .. .. 0: pte 0x0000000021fd501b pa 0x0000000087f54000
.. .. .. 1: pte 0x0000000021fd8c17 pa 0x0000000087f63000
.. .. .. 2: pte 0x0000000021fd8807 pa 0x0000000087f62000
.. .. .. 3: pte 0x0000000021fdb817 pa 0x0000000087f6e000
.. 255: pte 0x0000000021fd4801 pa 0x0000000087f52000
.. .. 511: pte 0x0000000021fd4c01 pa 0x0000000087f53000
.. .. .. 510: pte 0x0000000021fd0407 pa 0x0000000087f41000
.. .. .. 511: pte 0x0000000020001c0b pa 0x0000000080007000
Properties:inode number 26 dev 1 type 2 nlink 1 size 5090
Block number 0
Block content
CS22BTECH11050
Block number 1
Block content
CS22BTECH11050
Block number 2
Block content
CS22BTECH11050
Block number 3
Block content
CS22BTECH11050
Block number 4
Block content
CS22BTECH11050
$
```

ls command gives the following output.



Challenges faced

I have tried printing metadata for the files in the bmap() function by accessing the inode of the file But there were many issues which persisted.
bmap() takes struct inode and block number bn as parameters. Accessing this structure can give all the information about the file.
There are two conditions in this function

```
if(bn < NDIRECT){
  if((addr = ip->addrs[bn]) == 0){
    addr = balloc(ip->dev);
    if(addr == 0)
      return 0;
    ip->addrs[bn] = addr;
  }
 // printf("inode contents: number %d dev %d ref %d validity %d type %d major %d minor %d nlink %d size %d\n block number %d\n ",ip->inum,ip->dev,ip->ref,ip->valid
  return addr;
}
bn -= NDIRECT;

if(bn < NINDIRECT){
  // Load indirect block, allocating if necessary.
  if((addr = ip->addrs[NDIRECT]) == 0){
    addr = balloc(ip->dev);
    if(addr == 0)
      return 0;
    ip->addrs[NDIRECT] = addr;
  }
  bp = bread(ip->dev, addr);
  a = (uint*)bp->data;
  if((addr = a[bn]) == 0){
    addr = balloc(ip->dev);
    if(addr){
      a[bn] = addr;
      log_write(bp);
    }
  }
  brelse(bp);
 // printf("inode contents: number %d dev %d ref %d validity %d type %d major %d minor %d nlink %d size %d\n block number %d\n ",ip->inum,ip->dev,ip->ref,ip->valid
  return addr;
}

panic("bmap: out of range");
}
```

If the commented print statement in the second condition is uncommented then there isn't any problem and the details are getting printed for every block allocated to the file
But if the commented print statement of first if is uncommented then there are multiple prints with inode number as 1 and block number as 0.
This is observed when any other program is run as well(not just lab4_files).
The same is printed just when make quem is called. I think they might be the blocks corresponding to init process.
If this print is not uncommented and only the second print is used then the details of files with smaller size aren't getting printed.
Here is a screenshot of the output I got by placing both prints.
As this isn't giving the correct output I have opted for the above method of accessing inode number from stat and printing block numbers from it.

```
.. .. .. 3: pte 0x0000000021fd5c07 pa 0x0000000087f57000
.. .. .. 4: pte 0x0000000021fd5817 pa 0x0000000087f56000
.. 255: pte 0x0000000021fd7801 pa 0x0000000087f5e000
.. .. 511: pte 0x0000000021fd7401 pa 0x0000000087f5d000
.. .. .. 510: pte 0x0000000021fdb407 pa 0x0000000087f6d000
.. .. .. 511: pte 0x0000000020001c0b pa 0x0000000080007000
inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 inode contents: number 1 dev 1 ref 3 validity 1 type 1 major 0 minor 0 nlink 1 size 1024
 block number 0
 $
```

Ln 418, Col 4 Spaces: 2 UTF-8 LF {} C Go Live Mac Prettier