Operating Systems CS-450 Assignment #4 (Group)

Sivasenthil Namachivayan
A20391478
Sravan Kumar Allu
A20343634

April 26,2017

Changes made in the Code:

- In stat.h, we have added the T_SFILE and we have defined with number#4. This will help us in knowing that the file being accessed is the small one.
- We added a new flag for open in the file fcntl.h as shown below

  ```
  #define O_SFILE 0x400
  ```
- In sysfile.c, we made changes to the create () function. We added below code to include the smallfile types.

  ```
     if(type == T_SFILE && ip->type == T_SFILE)
    {
                return ip;
     }
  ```
- We have also modified system call sys_open() and we have added functionality to create either smallfile type if it is less than 52 bytes or regular file. Previously we have only functionality of creating only a regular file. Below are the changes that are made.

  ```
   if(omode & O_CREATE)
  {
          if (omode & O_SFILE)
          {
          // creates a SFILE
          if((ip = create(path, T_SFILE, 0, 0)) == 0)
                  return -1;
          }
          else
          {
                  // creates a REGULAR file
                   if((ip = create(path, T_FILE, 0, 0)) == 0)
                  return -1;
          }
  }
  ```
- In fs.c we have modified the iput(), readi() and writei() function. We have made changes and below is the full code of the function after modification.

  ```
  void iput(struct inode *ip)
  {
          acquire(&icache.lock);
          if(ip->ref == 1 && (ip->flags & I_VALID) && ip->nlink == 0){
          // inode is no longer used: truncate and free inode.
          if(ip->flags & I_BUSY)
          panic("iput busy");
          ip->flags |= I_BUSY;
          release(&icache.lock);
          // Don't need to free block if it is a small file
          if (ip->type != T_SFILE) {
  ```

```
                        itrunc(ip);
                        }
            ip->type = 0;
            iupdate(ip);
            acquire(&icache.lock);
            ip->flags = 0;
            wakeup(ip);
            }
ip->ref--;
release(&icache.lock);
}
```

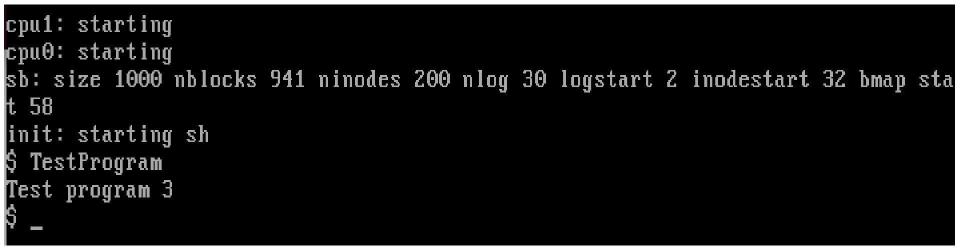
Below is the modified code for the readi() function.

```
int readi(struct inode *ip, char *dst, uint off, uint n)
{
  uint tot, m;
  struct buf *bp;

  if(ip->type == T_DEV)
  {
    if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].read)
      return -1;
    return devsw[ip->major].read(ip, dst, n);
  }

  if(off > ip->size || off + n < off)
    return -1;
  if(off + n > ip->size)
    n = ip->size - off;

  // 2 cases
  // handle T_SFILE
  if (ip->type == T_SFILE) {
    memmove(dst, (char*)(ip->addrs) + off, n);
  } else {
    // handle T_FILE
    for(tot=0; tot<n; tot+=m, off+=m, dst+=m){
      uint sector_number = bmap(ip, off/BSIZE);
      if(sector_number == 0){ //failed to find block
        panic("readi: trying to read a block that was never allocated");
      }

      bp = bread(ip->dev, sector_number);
      m = min(n - tot, BSIZE - off%BSIZE);
      memmove(dst, bp->data + off%BSIZE, m);
      brelse(bp);
    }
  }

  return n;
}
```

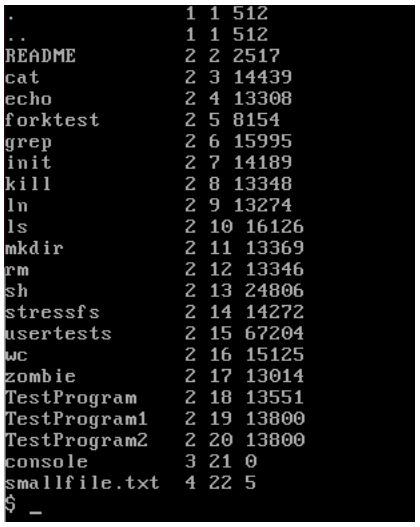
we have also changed writei() function and below is the code for the same.

```
int
writei(struct inode *ip, char *src, uint off, uint n)
{
  uint tot, m;
  struct buf *bp;

  if(ip->type == T_DEV){
    if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].write)
      return -1;
    return devsw[ip->major].write(ip, src, n);
  }

  if(off > ip->size || off + n < off)
    return -1;
  if(off + n > MAXFILE*BSIZE)
    n = MAXFILE*BSIZE - off;
  // try to make the small file bigger than limit
  if(ip->type == T_SFILE && off + n > (NDIRECT + 1) * sizeof(uint))
    n = (NDIRECT + 1) * sizeof(uint) - off;

  // 2 cases
  // handle T_SFILE
  if (ip->type == T_SFILE) {
    memmove((char*)(ip->addrs) + off, src, n);
    off += n;
  } else {
    // handle T_FILE
    for(tot=0; tot<n; tot+=m, off+=m, src+=m){
      uint sector_number = bmap(ip, off/BSIZE);
      if(sector_number == 0){ //failed to find block
        n = tot; //return number of bytes written so far
        break;
      }

      bp = bread(ip->dev, sector_number);
      m = min(n - tot, BSIZE - off%BSIZE);
      memmove(bp->data + off%BSIZE, src, m);
      bwrite(bp);
      brelse(bp);
    }
  }

  // If SMALLFILE, must update inode
  if(ip->type == T_SFILE || (n > 0 && off > ip->size)) {
    if (n > 0 && off > ip->size) {
      ip->size = off;
    }
    iupdate(ip);
  }
  return n;
}
```

Test Cases:
There are 3 test cases involved in this execution.
Case 1: The total size of the file is below 52 bytes (Maximum bytes that an inode can contain in the data region)
1.0

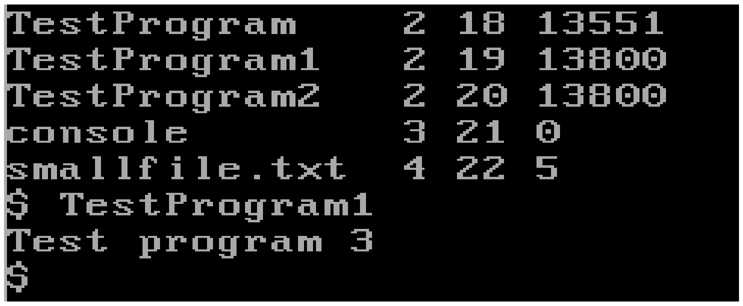

```
cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta
t 58
init: starting sh
$ TestProgram
Test program 3
$ _
```

1.1


```
.              1 1 512
..             1 1 512
README         2 2 2517
cat            2 3 14439
echo           2 4 13308
forktest       2 5 8154
grep           2 6 15995
init           2 7 14189
kill           2 8 13348
ln             2 9 13274
ls             2 10 16126
mkdir          2 11 13369
rm             2 12 13346
sh             2 13 24806
stressfs       2 14 14272
usertests      2 15 67204
wc             2 16 15125
zombie         2 17 13014
TestProgram    2 18 13551
TestProgram1   2 19 13800
TestProgram2   2 20 13800
console        3 21 0
smallfile.txt  4 22 5
$ _
```

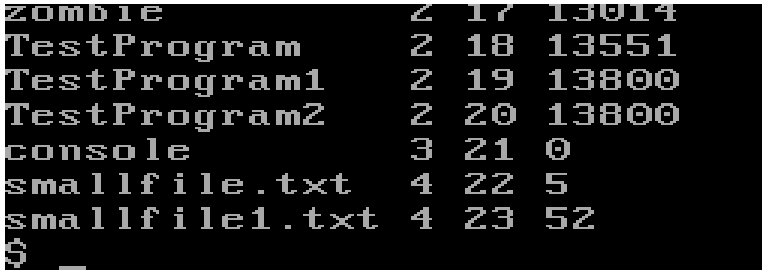
In the above screenshot 1.0 the User program name is TestProgram.C. You can see that the smallfile.txt is created in the 1.1 screenshot. The file type is of 4- which denotes Small file which can be accomodated in the inode. The total byte of this file is 5 bytes.

Case 2: The total size of the file is equal to 52 bytes (Maximum bytes that an inode can contain in the data region)
2.0



```
TestProgram     2 18 13551
TestProgram1    2 19 13800
TestProgram2    2 20 13800
console         3 21 0
smallfile.txt   4 22 5
$ TestProgram1
Test program 3
$
```

2.1

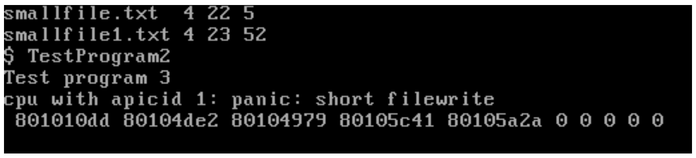


```
zombie          2 17 13014
TestProgram     2 18 13551
TestProgram1    2 19 13800
TestProgram2    2 20 13800
console         3 21 0
smallfile.txt   4 22 5
smallfile1.txt  4 23 52
$ _
```

In the above screenshot 2.0 the User program name is TestProgram1.C. You can see that the smallfile1.txt is created in the 2.1 screenshot. The file type is of 4- which denotes Small file which can be accomodated in the inode. The total byte of this file is 52 bytes.

Case 3: The total size of the file is greater than 52 bytes

3.0



```
smallfile.txt  4 22 5
smallfile1.txt 4 23 52
$ TestProgram2
Test program 3
cpu with apicid 1: panic: short filewrite
 801010dd 80104de2 80104979 80105c41 80105a2a 0 0 0 0 0
```

In the above screenshot 3.0 the User program name is TestProgram2.C. You can see that no file is created. We got a panic error which shows that the file cannot be written.