Ran Huo A20437439
Yin Jiang A20250197

Design

In this programming assignment, we implement a method that improves the overhead in handling small files in xv6.

Inode structure can be found in fs.h. Thus, for files smaller than 13*4=52 bytes, data store directly in the inode addr[NDIRECT+1].

```
// On-disk inode structure
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEV only)
  short minor;          // Minor device number (T_DEV only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+1];   // Data block addresses
};
```

Step 1)

First we create a new types of files and directories in stat.h

```
#define T_SMALLFILE 5 //Small File
#define T_SFDIR 4  // Directory for small files
```

Step 2)

Create new system call mkSFdir.

Files created under small files directories are expected to be small and fits into an inode. In syscall.c, add following lines

```
//new mkSFdir
extern int sys_mkSFdir(void);
```

```
[SYS_mkSFdir] sys_mkSFdir,
```

In usys.s

Add SYSCALL(mkSFdir)

In syscall.h, add

```
#define SYS_mkSFdir 22
```

In sysfile.c

```

//system call for mkSFdir
int
sys_mkSFdir(void)
{
  char *path;
```

```
  struct inode *ip;

  begin_op();
  if(argstr(0, &path) < 0 || (ip = create(path, T_SFDIR, 0, 0)) == 0){
    end_op();
    return -1;
  }
  iunlockput(ip);
  end_op();
  return 0;
}
```

In user.h

```
int mkSFdir(const char*);
```

create mkSFdir.c like mkdir.c

```
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  int i;

  if(argc < 2){
    printf(2, "Usage: mkSFdir files...\n");
    exit();
  }

  for(i = 1; i < argc; i++){
    if(mkSFdir(argv[i]) < 0){
      printf(2, "mkSFdir: %s failed to create\n", argv[i]);
      break;
    }
  }

  exit();
}
```

Step 3)

Modify open(), read(), write() system call.

Modify sys_open in sysfile.c. Check the file directory before creating new files. If the type is T_SFDIR, then the new file type is small file, otherwise, create regular file.

```
//find file directory
  int n=sizeof(path);
  int i,j;
  char parent[20];

  for(i=n-1; i>=0; i--){
     if(path[i]=='/'){
      break;
     }
  }
  parent[i] = '\0';
  for(j=0;j<i;j++)
    parent[j]=path[j];
  //create file depends on its directory type
  ip=namei(parent);

  if(omode & O_CREATE){
    if(ip->type==T_SFDIR){
      ip=create(path, T_SMALLFILE,0,0);
    }else{
      ip = create(path, T_FILE, 0, 0);
    }
```

Modify int readi(struct inode *ip, char *dst, uint off, uint n) in fs.c
If file type is T_SMALLFILE, read data from inode, otherwise, read from data blocks as normal file.

```
//if file type is T_SMALLFILE, read from inode
  if(ip->type == T_SMALLFILE){
     memmove(dst, (char*)(ip->addrs)+off,n);
  }else
```

Modify int writei(struct inode *ip, char *dst, uint off, uint n) in fs.c

1) When type is T_SMALLFILE and the length is less than 52, write the data into addr[].
2) When type is T_SMALLFILE and the length is greater than 52, overflow happens, convert the file type to T_FILE, write into the data blocks.
3) When type is T_FILE, write into data blocks directly.
4) To convert small files to normal file, we add function void toNormal(struct inode *ip), in the function, context in addr[] is saved into the data blocks, and the addr[] is reset to 0.

```
//write into the inode when the file type is T_SMALLFILE and the length is within 52
  if(ip->type==T_SMALLFILE && off+n<=52){
     memmove((uchar*)ip->addrs+off,src,n);
     off+=n;
  }else{
```

```
  //when overflow happens, convert small file to regular file type
    if(ip->type==T_SMALLFILE && off+n>52){
      cprintf("Buffer is big...\n");
      cprintf("converting small file to normal file...\n");
      //ip->type=T_FILE;
      toNormal(ip);
      for(tot=0; tot<n; tot+=m, off+=m, src+=m){
        bp = bread(ip->dev, bmap(ip, off/BSIZE));
        m = min(n - tot, BSIZE - off%BSIZE);
        memmove(bp->data + off%BSIZE, src, m);
        log_write(bp);
        brelse(bp);
       }
    }else{
      for(tot=0; tot<n; tot+=m, off+=m, src+=m){
        bp = bread(ip->dev, bmap(ip, off/BSIZE));
        m = min(n - tot, BSIZE - off%BSIZE);
        memmove(bp->data + off%BSIZE, src, m);
        log_write(bp);
        brelse(bp);
       }
     }
  }
```

```
//convert small file to normal file
void
toNormal(struct inode *ip) {
  char buff[ip->size];
  struct buf *bp;
  if(ip->type == T_FILE) {
    return;
  }
  ip->type = T_FILE;
  //copy the context from the addrs to buff
  memmove(buff, (uchar*)ip->addrs, ip->size);
  //clean the addrs
  memset(ip->addrs, 0, ip->size * sizeof(int));
  //allocate data blocks
  bp= bread(ip->dev, bmap(ip, 0));
  //write buff into data blocks
  memmove(bp->data, buff, ip->size);
  log_write(bp);
  brelse(bp);

  iupdate(ip);
  return;
}
```

Since small file doesn't need any data blocks. In bitmap function in sysfile.c, check the file type before allocate data blocks to files.

```
//return if the file type is T_SMALLFILE
  if(ip->type==T_SMALLFILE)
    return -1;
```

Step 4)

Since we expect files created under small file directories to be small files, linking small files to normal file directories is not allowed. Thus, we have normal files in normal directories, small files and normal files converted from small files in small file directories. If the linking is allowed and we still want to keep such a file system organization, we can design a small file link (symbolic link) that stores the pathname of the linked-to small file. Then, we can link small files to normal directories.

Testing:

1) Create a small file directory

```
mkSFdir("foo");
```

2) Create a small file under small file directory

```
printf(1, "Test creating small file\n");
if((fd = open("foo/test_file1.txt", O_CREATE | O_RDWR)) < 0){
   printf(1, "Failed to create the file\n");
} else {

    printf(1, "Successfully create the file\n");
    char * fileName = "foo/test_file1.txt";
    show(fileName);
    struct stat sb;
    stat("foo/test_file1.txt", &sb);
    if(sb.type==T_SMALLFILE)
      printf(1,"small file\n");
    else
      printf(1,"file");
}
close(fd);
```

```
---------------------------
Test creating small file
Successfully create the file
TYPE: T_SMALLFILE
FILENAME: foo/test_file1.txt
        -INODE_NUMBER: 22
        -Size: 0
        -nlink: 1
---------------------------
```

A small has been created with inode_number 22

3) Create a normal file under normal file directory

```
-------------------------
Testing link normal file to normal directory
Successfully create the file
TYPE: T_FILE
FILENAME: normalFile.txt
        -INODE_NUMBER: 25
        -Size: 0
        -nlink: 1
Successfully linked the file
TYPE: T_FILE
FILENAME: normallink.txt
        -INODE_NUMBER: 25
        -Size: 0
        -nlink: 2
-------------------------
```

A normal file has been created and will be linked to a normal directory.

4) Write a small buffer to a small file

```
---------------------------
Test write small buffer to a small file
Successfully create the file
TYPE: T_SMALLFILE
FILENAME: foo/test_file2.txt
        -INODE_NUMBER: 23
        -Size: 3
        -nlink: 1
```

Write 3 characters to a small file, size becomes to be 3.

5) Write a big buffer to a small file.

```
printf(1, "Test write overflow a small file and convert to normal file\n");
 if((fd = open("foo/test_file2.txt", O_CREATE | O_RDWR)) < 0){
    printf(1, "Failed to create the file\n");
 } else {
    printf(1, "Successfully create the file\n");

   int n;
    if((n = write(fd, &buff, 63)) != 63) {
     printf(1, "Write Failed\n");

   } else {
 close(fd);
      char * fileName = "foo/test_file2.txt";
```

```
    show(fileName);
  }
```

```
Test write overflow a small file and convert to normal file
Writing more...
Buffer is too big...
converting small file to normal file...
TYPE: T_FILE
FILENAME: foo/test_file2.txt
        -INODE_NUMBER: 23
        -Size: 66
        -nlink: 1
```

Write a big buffer to a small file. Convert the small file to normal file. Inode_number is still 23. Type is normal file (T_FILE). Size is 66.

6) Link a small file to a small file directory

```
printf(1, "Testing link small file to small directory \n");
 if ((fd = open("foo/test_file1.txt", O_RDWR)) < 0){
   printf(1, "Failed to open the file\n");
 } else {
   printf(1, "Successfully create the file\n");

   if (link("foo/test_file1.txt", "foo/smalllink.txt") < 0){
     printf(1, "Failed to link the file\n");
   } else {
     printf(1, "Successfully linked the file\n");
     char * fileName = "foo/smalllink.txt";
     show(fileName);
     close(fd);
     }
 }
```

```
---------------------------
Testing link small file to small directory
Successfully create the file
Successfully linked the file
TYPE: T_SMALLFILE
FILENAME: foo/smalllink.txt
        -INODE_NUMBER: 22
        -Size: 0
        -nlink: 2
```

A small file is linked to a small file directory. Nlink is 2.

7) Write to a small file using the link

```
printf(1, "Writing to small file using the link...\n");
 if((fd = open("foo/smalllink.txt", O_RDWR)) < 0){
    printf(1, "Failed to open the file\n");
 } else {
    printf(1, "Successfully open the file\n");
    char buff0[3] = "012";
    int n;
    if((n = write(fd, &buff0, 3)) != 3) {
```

```
    printf(1, "Write Failed\n");
   } else {
    close(fd);
    char * fileName = "foo/smalllink.txt";
    show(fileName);
   }
}
```

```
Writing to small file using the link...
Successfully open the file
TYPE: T_SMALLFILE
FILENAME: foo/smalllink.txt
        -INODE_NUMBER: 22
        -Size: 3
        -nlink: 2
--------------------------
```

We can use the link to write to the same file. Inode_number is 22. Size is 3.

8) Link a small file to a normal file directory

```
///testing normal link to small
printf(1, "Testing link small file to normal directory\n");
if ((fd = open("foo/test_file1.txt", O_RDWR)) < 0){
  printf(1, "Failed to open the file\n");
} else {
  printf(1, "Successfully create the file\n");

  if (link("foo/test_file1.txt", "smalllink1.txt") < 0){
    printf(1, "Failed to link the file\n");
  } else {
    printf(1, "Successfully linked the file\n");
    char * fileName = "smalllink1.txt";
    show(fileName);
    close(fd);
  }
}
```

```
--------------------------
Testing link small file to normal directory
Successfully create the file
TYPE: T_SMALLFILE
FILENAME: foo/smalllink1.txt
        -INODE_NUMBER: 24
        -Size: 0
        -nlink: 1
Failed to link the file
--------------------------
```

Link a small file to a normal file directory is not allowed. “Failed to link the file”

9) Link a normal file to a normal file directory

```
printf(1, "Testing link normal file to normal directory\n");
if ((fd = open("normalFile.txt", O_CREATE | O_RDWR)) < 0){
  printf(1, "Failed to create the file\n");
} else {
```

```
  printf(1, "Successfully create the file\n");
  char * fileName = "normalFile.txt";
  show(fileName);
  if (link("normalFile.txt", "normallink1.txt") < 0){
    printf(1, "Failed to link the file\n");
  } else {
    printf(1, "Successfully linked the file\n");
    char * fileName = "normallink1.txt";
    show(fileName);
    close(fd);
  }
}
```

```
---------------------------
Testing link normal file to normal directory
Successfully create the file
TYPE: T_FILE
FILENAME: normalFile.txt
        -INODE_NUMBER: 25
        -Size: 0
        -nlink: 1
Successfully linked the file
TYPE: T_FILE
FILENAME: normallink.txt
        -INODE_NUMBER: 25
        -Size: 0
        -nlink: 2
---------------------------
```

Link a normal file to a normal file directory is successful.