Adding Small File Handling Mechanism in Xv6 OS

- **Name : Nithin Rao**
- **CWID : A20446755**
- **Name : Varun Gunda**
- **CWID : A20453991**

Idea

The on-disk inode "dinode" defined in fs.h and the "inode" structure defined in file.h have "addrs" member to store the addresses of the disk blocks where the data will be stored. This addrs has 13 elements out of which 12 are direct block pointers and 1 is indirect block pointer. Hence, for small files, instead of allocating disk blocks and storing the content in the disk block, we store the data in the memory pointed by "addrs". This addrs occupies 52 bytes of memory since there are 13 elements of size 4 bytes each. We used this 52 bytes to store the contents of small file.

Files Changed

Modifications are made to the following files for handling small files:

1. stat.h
2. fs.c
3. user.h
4. usys.S
5. syscall.c
6. syscall.h
7. sysfile.c
8. mkSFdir.c (New file)
9. Makefile
10. defs.h

Apart from these changes, the following files are modified/created to support handling small files:

1. touch.c: Implements "touch" command to create files in linux style.
2. console.c: Modified this to support printing characters on the console
3. ls.c
4. exec.c

Modified ls.c and exec.c to support execution of commands within directories other than root.

Creation of Small Directory

A new directory creation command "mkSFdir" is created to help creating small file directories. Files "syscall.c", "syscall.h", "usys.S", "user.h", "Makefile" are modified accordingly (like in previous assignments) to

add this new command.

The code in mkSFDir.c is as shown below:

```
#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();
}
```

This command is similar to mkdir.c but this calls the system call "mksfdir" to create the small directory. The code for this system call "sys_mksfdir" is shown below:

```
int sys_mksfdir(void)
{
  return mkdir(T_SDIR);
}
```

This calls the mkdir function to create directory of type T_SDIR. The sys_mkdir is also modified to call this mkdir function with type T_DIR. The code is as shown below:

```
int sys_mkdir(void)
{
  return mkdir(T_DIR);
}
```

```
static int mkdir(short dir_type)
{

  char *path;
  struct inode *ip;

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

The constants T_SDIR & T_SFILE which will be used to create small files are defined in stat.h.

```
--- ./original/stat.h   2019-09-24 13:45:16.000000000 -0500
+++ ./changed/stat.h    2019-11-26 07:57:07.000000000 -0600
@@ -1,7 +1,8 @@
 #define T_DIR  1   // Directory
 #define T_FILE 2   // File
 #define T_DEV  3   // Device
-
+#define T_SDIR  4   // Small Directory
+#define T_SFILE 5   // Small File
 struct stat {
   short type;  // Type of file
   int dev;     // File system's disk device
```

Creation of Small File

Major modifications are made to fs.c, fs.h to suppor the creation of small files. Two new functions are introduced in fs.h which check if the input argument is a directory or a file.

```
diff -uw ./original/defs.h ./changed/defs.h
--- ./original/defs.h   2019-09-24 13:45:16.000000000 -0500
+++ ./changed/defs.h    2019-11-26 07:57:07.000000000 -0600
@@ -35,6 +35,8 @@
 int             filewrite(struct file*, char*, int n);

 // fs.c
+int             is_type_dir(short type);
+int             is_type_file(short type);
```

```
void            readsb(int dev, struct superblock *sb);
int             dirlink(struct inode*, char*, uint);
struct inode*   dirlookup(struct inode*, char*, uint*);
```

The file fs.c is modified as shown below. The flow is as follows: The user program executes the read command on a small file created in the small directory (By default if a file is being created in a small directory, it is considered as a small file). The function readi in fs.c checks if the file is a small file. If the file is a small file, then content stored at "addrs" location is read. Else, the content is read from the blocks pointed by addrs which is the default way.

Similarly, when a user program executes the write command on a small file, the writei function in fs.c checks if the file is a small file. If the file is small file, content is written to the memory allocated to addrs. This function also checks if the size of the content to be written exceeds 52 characters and if it exceeds, this function calls convert_to_normal_file function which modifies this inode to be a normal file inode.

There are also small modifications made in some functions which look for type T_DIR or T_FILE to also look for T_SDIR and T_SFILE and provide panic messages accordingly.

```
diff -uw ./original/fs.c ./changed/fs.c
--- ./original/fs.c     2019-09-24 13:45:16.000000000 -0500
+++ ./changed/fs.c      2019-11-26 07:57:07.000000000 -0600
@@ -21,6 +21,24 @@
 #include "buf.h"
 #include "file.h"

//Check if the argument type is directory or small directory
+int is_type_dir(short type)
+{
+  if (type == T_DIR || type == T_SDIR)
+  {
+    return 1;
+  }
+  return 0;
+}

//Check if the argument type is file  or small file
+int is_type_file(short type)
+{
+  if (type == T_FILE || type == T_SFILE)
+  {
+    return 1;
+  }
+  return 0;
+}
+
 #define min(a, b) ((a) < (b) ? (a) : (b))
 static void itrunc(struct inode*);
 // there should be one superblock per disk device, but we run with
```

```
@@ -466,25 +496,55 @@
   if(off + n > ip->size)
     n = ip->size - off;
//If type of file is small file, then copy the content directly to memory
pointed by addrs. Else use disck blocks to store content
-  for(tot=0; tot<n; tot+=m, off+=m, dst+=m){
+  if (ip->type == T_SFILE)
+  {
+    memmove(dst, (char*)(ip->addrs) + off, n);
+  }
+  else
+  {
+    for (tot = 0; tot < n; tot += m, off += m, dst += m)
+    {
     bp = bread(ip->dev, bmap(ip, off/BSIZE));
     m = min(n - tot, BSIZE - off%BSIZE);
     memmove(dst, bp->data + off%BSIZE, m);
     brelse(bp);
   }
+  }
   return n;
 }

//Convert the given small file inode to normal file inode
+static void convert_to_normal_file(struct inode *ip){
+  uint tot, m, off = 0;
+  struct buf *bp;
+
+  if(ip->size > 0){
+    char src_data[ip->size];
+    char * src = src_data;
+    strncpy(src, (char*)ip->addrs, ip->size);
+    memset((char *)(ip->addrs), 0, sizeof(uint) * (NDIRECT + 1));
+    for (tot = 0; tot < ip->size; tot += m, off += m, src += m)
+    {
+      bp = bread(ip->dev, bmap(ip, off / BSIZE));
+      m = min(ip->size - tot, BSIZE - off % BSIZE);
+      memmove(bp->data + off % BSIZE, src, m);
+      log_write(bp);
+      brelse(bp);
+    }
+  }
+  ip->type = T_FILE;
+}
+
 // PAGEBREAK!
 // Write data to inode.
 // Caller must hold ip->lock.
@@ -495,16 +555,37 @@
   if(off + n > MAXFILE*BSIZE)
     return -1;

  for(tot=0; tot<n; tot+=m, off+=m, src+=m){
```

```
//If the size of the content is beyond 52, then convert the small file to
normal file
+  if (ip->type == T_SFILE && off + n > sizeof(uint) * (NDIRECT + 1)){
+    convert_to_normal_file(ip);
+  }
+
+  if (ip->type == T_SFILE)
+  {
+    memmove((char *)(ip->addrs) + off, src, n);
+    off += n;
+  }
+  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);
   }
+  }

-  if(n > 0 && off > ip->size){
+  if ((n > 0 && off > ip->size) || ip->type == T_SFILE)
+  {
+    if (n > 0 && off > ip->size)
+    {
     ip->size = off;
+    }
     iupdate(ip);
   }
   return n;
@@ -527,15 +607,17 @@
   uint off, inum;
   struct dirent de;

-  if(dp->type != T_DIR)
+  if (!is_type_dir(dp->type))
     panic("dirlookup not DIR");

@@ -631,25 +715,30 @@
   else
     ip = idup(myproc()->cwd);

  while ((path = skipelem(path, name)) != 0)
+  {
     ilock(ip);
-    if(ip->type != T_DIR){
+    if (!is_type_dir(ip->type))
+    {
       iunlockput(ip);
       return 0;
     }
```

Additional changes to XV6 OS

New command "touch" is added to create files in linux style. This command creates a file in the given path. The code of "touch.c" is shown below:

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

int
main(int argc, char *argv[])
{
  int i;
  if(argc < 2){
    printf(2, "Usage: touch files...\n");
    exit();
  }
  for(i = 1; i < argc; i++){
    if(open(argv[i], O_CREATE | O_RDWR) < 0){
      printf(2, "touch: failed to create file %s\n", argv[i]);
      break;
    }
  }
  exit();
}
```

Support is added to console.c to print a character using "%c". The modification made in console.c is as show below:

```
--- ./original/console.c        2019-09-24 13:45:16.000000000 -0500
+++ ./changed/console.c 2019-11-26 07:57:07.000000000 -0600
@@ -19,7 +19,8 @@

  for(i = 0; (c = fmt[i] & 0xff) != 0; i++){
    if(c != '%'){
       consputc(c);
       continue;
     }
     c = fmt[++i] & 0xff;
     if(c == 0)
       break;
+    switch (c)
+    {
     case 'd':
       printint(*argp++, 10, 1);
```

```
       break;
+    case 'c':
+      consputc(*argp++);
+      break;
     case 'x':
     case 'p':
       printint(*argp++, 16, 0);
@@ -103,8 +110,7 @@
     release(&cons.lock);
 }
-
```

The command "ls" is modified to handle the case of displaying the contents of small directories:

```
diff -uw ./original/ls.c ./changed/ls.c
--- ./original/ls.c     2019-09-24 13:45:16.000000000 -0500
+++ ./changed/ls.c      2019-11-26 07:57:07.000000000 -0600
@@ -47,6 +47,7 @@
     break;

   case T_DIR:
+  case T_SDIR:
     if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){
       printf(1, "ls: path too long\n");
       break;
```

The command "exec" is modified to support execution of commands inside any of the directories.

```
diff -uw ./original/exec.c ./changed/exec.c
--- ./original/exec.c   2019-09-24 13:45:16.000000000 -0500
+++ ./changed/exec.c    2019-11-26 07:57:07.000000000 -0600
@@ -7,10 +7,35 @@
 #include "x86.h"
 #include "elf.h"

+char*
+strcpy(char *s, const char *t)
+{
+  char *os;
+
+  os = s;
+  while((*s++ = *t++) != 0)
+    ;
+  return os;
+}
+
+char*
+strcat(char *s, const char *t)
+{
```

```
+  return strcpy(s + strlen(s), t);
+}
+
+char * path_from_parent(char * path){
+    char * path_from_parent = kalloc();// malloc(sizeof(char) *
(strlen(path) + 2));
+    strcpy(path_from_parent, "/");
+    strcat(path_from_parent, path);
+    return path_from_parent;
+}
+
 int
 exec(char *path, char **argv)
 {
   char *s, *last;
+  char *cur_path = path;
   int i, off;
   uint argc, sz, sp, ustack[3+MAXARG+1];
   struct elfhdr elf;
@@ -21,11 +46,17 @@

   begin_op();

-  if((ip = namei(path)) == 0){
+  if((ip = namei(cur_path)) == 0){
+    char * path_wrt_parent = path_from_parent(cur_path);
+    if((ip = namei(path_wrt_parent)) == 0){
     end_op();
     cprintf("exec: fail\n");
     return -1;
   }
+    cur_path = path_wrt_parent;
+  }
   ilock(ip);
   pgdir = 0;
```

Linking a Small File to other Directory

In this current design, where the read, write functions etc., work based on the file type in the inode, we can easily link any small file to non-small directory. The output when linking the small file to non-small directory is as shown below:

```
$ mkdir nordir
$ mkSFdir smalldir
$ touch smalldir/sfile
$ ln smalldir/sfile nordir/nfile
$ ls smalldir
.              4 23 48
..             1 1 512
sfile          5 24 0
$ ls nordir
```

```
.               1 22 48
..              1 1 512
nfile           5 24 0
```

Here we made a small directory and created a small file in it. Later we created a normal directory using mkdir and linked this small file to normal directory. As expected,since this is a hardlink, the samek inode number is copied to the normal directory "nordir" inode. Now, if open, read or write commands are called on this "nfile" the code will check for the type of file from the inode and based on that perform the read/write operations.