CSCI 4730/6730
(PA #3 – FS Simulator)

In Kee Kim

School of Computing

University of Georgia

PA #3 – EXT2-like File System

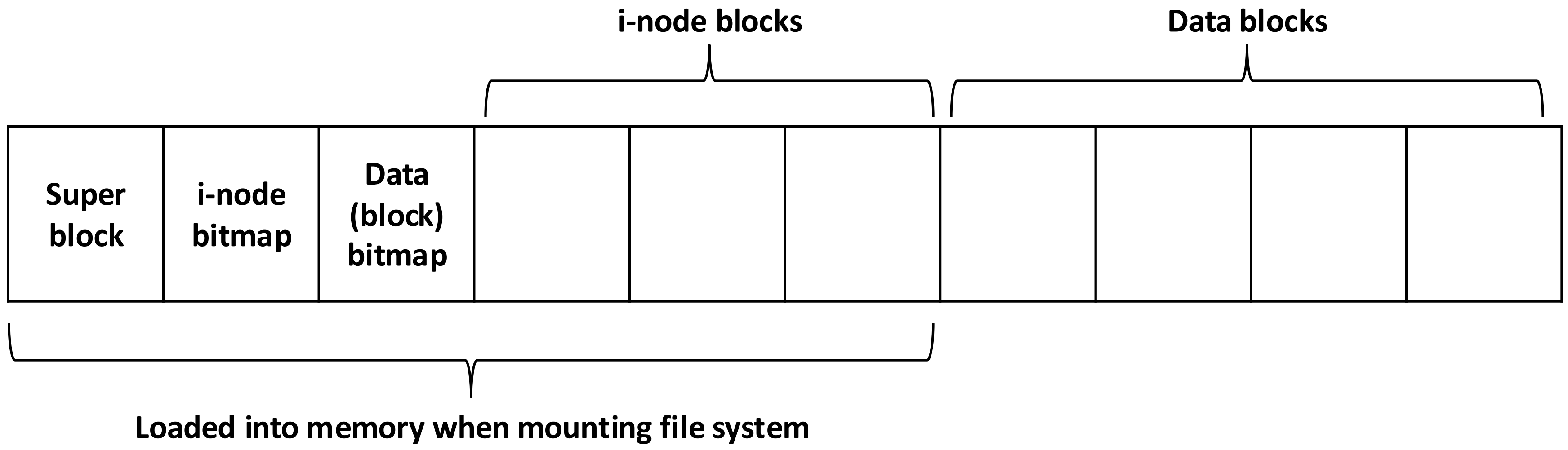
- Part 1: File management (70pt)
- Part 2: Directory management: (30pt)

PA #3 – Goal

- Implementing EXT2-like file system (no journaling) simulator to understand mechanism of modern Linux style file system operations

- The file system simulator should be able to
  1. Browse the disk information as well as file and directory list, and display meta data file and directory
     - `df, stat, ls`
  2. Create and delete files and directories, and
     - `create, rm, mkdir, rmdir`
  3. Read files.
     - `cat, read`

File System Structure in the Simulator



- Disk block size is 512 bytes
- Disk has 4096 blocks
- **Size of i-node entry is 128 bytes, 4 i-node entries can be stored in a single i-node block**
- Superblock, i-node bitmap, and data bitmap are loaded into memory when mounting FS

How to start simulator?

- **`$./fs_sim disk.dat`**
  - **`disk.dat`** is an argument, you can use any name for this

Part #1: File Management

- **70 points (easier part)**
- File System characteristics you need to know…
  - Only uses **direct blocks** in the i-node to support small size files
    - up to 6,144 bytes = 12 * 512 bytes, 12 blocks
    - all data should be in direct blocks
    - Max file size 6,144 bytes (defined in fs.h)
      - "`#define SMALL_FILE 6144`"
  - Each file in this FS has an i-node number
    - Unique # in a file system.
    - I-node structure (**Inode**) is defined in "**fs.h**" file.

Part #1: File Management Functions

Cmd	Args	Description	Func name in fs.c
`df`		Show file system information. e.g., # of free blocks, # of free i-nodes	**`fs_stat()`**
`create`	<name> <size>	Create a file with <filename> as a name in the current directory and fill it with size of random string.	**`file_create()`**
`stat`	<name>	Show the status (metadata) of the file or directory with name <name>. It displays • i-node information • file or directory • # of blocks allocated • Other info stored about this file/directory	**`file_stat()`**
`cat`	<name>	Print out the content of the file	**`file_cat()`**
`read`	<name> <offset> <size>	Read <size> bytes from the file <offset>	**`file_read()`** (Not implemented)
`rm`	<name>	Delete <name> file	**`file_remove()`** (Not implemented)
`ln`	<src_file> <new_file>	Create a hard link ("src_file" and "new_file" should be located in the same directory	**`hard_link()`** (Not implemented)

Where to start?

- fs_sim.c

```
int main(int argc, char **argv)
{
  ...
  fs_mount(argv[1]);  // file system mount
  ...
  while(fgets(input, 256, stdin)) {
    if(command(comm, "quit")) break;
    else if(command(comm, "exit")) break;
    else execute_command(comm, arg1, arg2, arg3, arg4, numArg - 1);
  }                                        // Command branch (e.g., df)
  ...
  fs_umount(argv[1]);  // file system unmount
}
```

Note: fs_mount, fs_unmount, execute_commands are in fs.c

fs_mount in fs.c

- Initialize Superblock (**`superBlock`** in fs.h), inodeMap, blockMap
  - **`inodeMap`**: i-node bitmap, **`blockMap`**: block bitmap

```
// Init file system superblock, inodeMap and blockMap
superBlock.magicNumber = MAGIC_NUMBER;
superBlock.freeBlockCount = MAX_BLOCK - (1+1+1+numInodeBlock);
superBlock.freeInodeCount = MAX_INODE;

//Init inodeMap
for(i = 0; i < MAX_INODE / 8; i++)
{
        // all inode is zero
        set_bit(inodeMap, i, 0);
}
//Init blockMap
for(i = 0; i < MAX_BLOCK / 8; i++)
{
        // 1 for superblock + inode bmap + data bmap + inode blocks
        if(i < (1+1+1+numInodeBlock)) set_bit(blockMap, i, 1);
        else set_bit(blockMap, i, 0); // 0 for all other blocks
}
```

fs_mount in fs.c

- Data structure of **superblock**, **inodeMap**, **blockMap** in fs.h

```c
#define MAX_BLOCK 4096
#define MAX_INODE 512

typedef struct {
    int magicNumber;
    int freeBlockCount;
    int freeInodeCount;
    char padding[500];
} SuperBlock;


extern SuperBlock superBlock;
extern char inodeMap[MAX_INODE / 8];   // bitmap
extern char blockMap[MAX_BLOCK / 8];   // bitmap
```

fs_mount in fs.c

- Create a new i-node entry
- i-node structure (**`Inode`**) is defined in fs.h

```
typedef struct {
    TYPE type;
    int owner;
    int group;
    struct timeval lastAccess;
    struct timeval created;
    int size;
    int blockCount;
    int directBlock[12];
    int link_count; //for hardlink
    char padding[16];
} Inode; //128 byte
```

fs_mount in fs.c

- Create a new i-node entry
- i-node structure is defined in fs.h

```
...

//Init root dir
int rootInode = get_free_inode();
curDirBlock = get_free_block();

inode[rootInode].type = directory;
inode[rootInode].owner = 0; // for directory
inode[rootInode].group = 0; // for directory
gettimeofday(&(inode[rootInode].created), NULL);
gettimeofday(&(inode[rootInode].lastAccess), NULL);
inode[rootInode].size = 1;
inode[rootInode].blockCount = 1;
inode[rootInode].directBlock[0] = curDirBlock;

curDir.numEntry = 1;
strncpy(curDir.dentry[0].name, ".", 1);
curDir.dentry[0].name[1] = '\0';
curDir.dentry[0].inode = rootInode;
disk_write(curDirBlock, (char*)&curDir);
```

fs_mount in fs.c

- APIs (in fs_util.c)
  - `int get_free_inode()`
    - Find next available i-node
    - return: -1 no available i-node block, otherwise i-node number
  - `int get_free_block()`
    - Find next available data block
    - return: -1 no available data block, otherwise data block index

execute_command in fs.c

For Files
(Part #1)

For Dirs
(Part #2)

```
int execute_command(...)
{
  if(command(comm, "df")) {
    return fs_stat();
  } else if(command(comm, "create")) {
    return file_create(arg1, atoi(arg2));
  } else if(command(comm, "stat")) {
    return file_stat(arg1);
  } else if(command(comm, "cat")) {
    return file_cat(arg1);
  } else if(command(comm, "read")) {
    return file_read(arg1, atoi(arg2), atoi(arg3));
  } else if(command(comm, "rm")) {
    return file_remove(arg1);
  } else if(command(comm, "ln")) {
    return hard_link(arg1, arg2);
  } else if(command(comm, "ls"))  {
    return ls();
  } else if(command(comm, "mkdir")) {
    return dir_make(arg1);
  } else if(command(comm, "rmdir")) {
    return dir_remove(arg1);
  } else if(command(comm, "cd")) {
    return dir_change(arg1);
  } else
    // error return
  return 0;
}
```

You need to implement red-colored functions

Recommendation

- Please try to understand pre-implemented libraries.
  - fs_stat()
  - **file_create()**
  - file_stat()
  - file_cat()

file_create (char *name, int size)

1. Calculate data block size (size / BLOCK_SIZE)
2. Create a file with random context (use size)
3. Get i-node number for the file
4. Update i-node (e.g., this is `file’, who is owner, etc)
5. Add file info to current dir (`curDir.dentry’)
6. Disk write (data block write)
7. Update i-node one more time (create time, last access time, etc.)

file_create (char *name, int size)

```
int file_create(char *name, int size) {
  // validation check routine #1
  // 1. file size check:  small than max and not negative size
  // 2. check if there is same file name search_cur_dir()
  // 3. check if directory full

  // calculate data block size
  int numBlock = size / BLOCK_SIZE; // 1 block = 512 bytes
  if(size % BLOCK_SIZE > 0) numBlock++;
  // 4. check if data block is full
  // 5. check if i-node is full
  ...

  // create file context with random context
  char *tmp = (char*) malloc(sizeof(int) * size + 1);
  rand_string(tmp, size); // create random content
  printf("New File: %s\n", tmp);
```

file_create (char *name, int size)

- Obtain available i-node index and fill the i-node entry

```
typedef struct {
    TYPE type;
    int owner;
    int group;
    struct timeval lastAccess;
    struct timeval created;
    int size;
    int blockCount;
    int directBlock[12];
    int link_count; //for hardlink
    char padding[16];
} Inode; //128 byte
```

```
// get i-node # for this file
inodeNum = get_free_inode();
if(inodeNum < 0) {
  // error
}

// update i-node info
inode[inodeNum].type = file;
inode[inodeNum].owner = 1;  // pre-defined
inode[inodeNum].group = 2;  // pre-defined
gettimeofday(&(inode[inodeNum].created), NULL);
gettimeofday(&(inode[inodeNum].lastAccess), NULL);
inode[inodeNum].size = size;
inode[inodeNum].blockCount = numBlock;
inode[inodeNum].link_count = 1;
```

file_create (char *name, int size)

- Add new file into current directory entry

```
// add file info to curr directory
strncpy(curDir.dentry[curDir.numEntry].name
       , name, strlen(name));
curDir.dentry[curDir.numEntry].name[strlen(name)] = '\0';
curDir.dentry[curDir.numEntry].inode = inodeNum;

printf("curdir %s, name %s\n"
       , curDir.dentry[curDir.numEntry].name, name);
curDir.numEntry++;
```

Data structure for `curDir.dentry` (in fs.h)

```
typedef struct {
  int numEntry;
  DirectoryEntry
       dentry[MAX_DIR_ENTRY];
  char padding[4];
} Dentry;
```

```
typedef struct {
  char name[MAX_FILE_NAME];
  int inode;
} DirectoryEntry;
```

file_create (char *name, int size)

- Get/set data blocks, write disk

```
typedef struct {
    TYPE type;
    int owner;
    int group;
    struct timeval lastAccess;
    struct timeval created;
    int size;
    int blockCount;
    int directBlock[12];
    int link_count; //for hardlink
    char padding[16];
} Inode; //128 byte
```

```
for(i = 0; i < numBlock; i++)
 {
   int block = get_free_block();
   if(block == -1) {
     // error case
     return -1;
   }
   //set direct block
   inode[inodeNum].directBlock[i] = block;

   // disk write
   disk_write(block, tmp+(i*BLOCK_SIZE));
 }
```

file_create (char *name, int size)

- Update last access time

```
  gettimeofday(&(inode[curDir.dentry[0].inode].lastAccess)
              , NULL);

  printf("file created: %s, inode %d, size %d\n"
          , name, inodeNum, size);

  free(tmp);
  return 0;
}
```

```
typedef struct {
    TYPE type;
    int owner;
    int group;
    struct timeval lastAccess;
    struct timeval created;
    int size;
    int blockCount;
    int directBlock[12];
    int link_count; //for hardlink
    char padding[16];
} Inode; //128 byte
```

Your implementation for Part #1

- `int file_read (char *name, int offset, int size)`
- `int file_remove (char *name)`
- `hard_link (arg1, arg2);`

file_read (name, offset, size)

- `file_cat` will give you a lot of hints
- Should support **random read**
- How to get i-node number from file name?
  - Use int **`search_cur_dir`** (char ***`name`**) in fs.c, then check (refer to file_cat)
    - 1) if i-node # is valid, 2) if it is file or directory. 3) if size of read request is valid
- inode array's direct block has the file context
  - Use **`disk_read`**() in disk.c
  - **Do not directly printf block context. You should define a new local variable (with malloc) and use memcpy**
    - Please refer to file_cat()
    - TA and I will check this!

file_read (name, offset, size)

- Procedure
  1. Get inode number from search_cur_dir(name);
  2. Validation check & error handling (e.g., size, offset, type)
  3. Create buffer with malloc, last index = ‘\0’
  4. Decide read block and offset
     - With inode number, block index, check inode struct
  5. Disk read & memcpy, multiblock read
  6. Update i-node for last access time

file_remove ()

- Think about how to check valid request and invalid request
- How to handle link_count?
  - If link_count == 1 or if link_count > 1? How do these differ?
- How to adjust directory entry array (see Dentry in fs.h)?
  - curDir.dentry[i]
  - curDir.numEntry
- Clean up inode bitmap and data bitmap?
  - Use set_bit() in fs_util.c
- Superblock?
  - freeInodeCount, freeBlockCount
  - Please check get_free_inode(), get_free_block() in fs_util.c
  - Should be opposite of get_free_inode() and get_free_block()

file_remove ()

- Procedure
  1. Get inode number (of file you want to delete) from search_cur_dir(name);
  2. Validation check & error handling (e.g., size, offset, type)
  3. Adjust adjust directory entry array
  4. Update last update time for the curr directory
  5. If there are hard links? Can you delete the file?
  6. Free data blocks Dmap & superblock
  7. Free inode block InodeMap & superblock

int hard_link (char *src, char *dest)

- dest → src, so src: existing file, desc: new file (not existing in fs)
  - How to check? search_cur_dir(char *name)
- Available directory entry should be able to add new “desc”
  - There is MAX_DIR_ENTRY_BLOCK
- Both src and desc must point the same i-node
  - **`curDir.dentry`**
    - **`curDir.dentry[xxx].name`** ← dest’s name
    - **`curDir.dentry[xxx].inode`** ← src’s name
- How about link_count?
  - Increase!

Part #2: Directory Management

- 30 points
- Directory Characteristics
  - Each directory has up to **`MAX_DIR_ENTRY`**
    - Both files and sub-directories
  - First directory entry is "."
  - **Second directory entry is ".."**

Part #2: Directory Management Functions

Cmd	Args	Description	Func name in fs.c
`ls`		Show the content of the current directory	`ls()`
`cd`	<name>	Change the current directory to <name>	`dir_change()`
`mkdir`	<name>	Create a sub-directory <name> under the current directory	`dir_make()` (Not implemented)
`rmdir`	<name>	Remove the sub-directory <name>	`dir_remove()` (Not implemented)

Part #2: Directory Management Functions

- Your job is to implement
  - `dir_make (char* name)`
  - `dir_remove (char* name)`

dir_make (char* name)

- i-node initialization: refer to fs_mount in fs.c
  - Similar, but not the same

- Procedure
  1. Check if the request is valid
     1. Check: name, max_dir_entry, # of freeblock/inode in superblock, etc.
  2. Get available inode #, new direct block → see fs_mount()
  3. Initialize inode
  4. Set directory entry
  5. Init for “.” and “..” as it is a subdirectory

dir_remove (char* name)

- Opposite of dir_make()
- Need recursive call – dir_remove() if the directory has sub directories
- Hard link?
- File delete?

What to submit to eLC

- Submit a tarball file using the following command
  - `$tar czvf p3.tar.gz README.pdf Makefile *.c *.h`
- README file with:
  - 1) Your name
  - 2) List what you have done and how did you test them. So that you can be sure to receive credit for the parts you've done.
  - 3) Description of your design
- The TA and instructor will check if your implementation correctly handles exception cases. e.g., i-node number validation, wrong arguments, etc.
- All source files are needed to compile, run and test your code
  - Makefile, All source files, **Do not submit object or executable files**
- Your code should be compiled and run correctly in odin machine.
  - **No credit will be given if your code failed to compile with "make" in odin.cs.uga.edu (we will use your makefile).**

Note

- Deadline
  - **11:59 pm, 11/23/2025 (Sunday)**
- You need a thorough review of the existing code.
- The late submission policy will be applied.