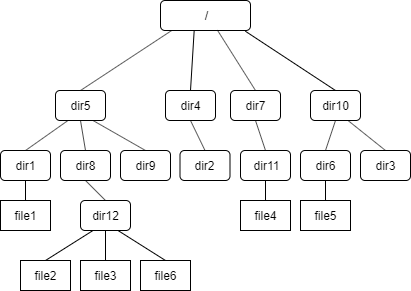
Test Cases

1. HD

The HD file represent the hard disk. The file system structure has been initialized properly on this hard disk. You do not need to do any modifications on HD.

The file system structure is shown below:



1. Recommend test cases:
2. open\_test()

You can call open\_t() to get the inode of a file, for example :

char pathname[MAX\_COMMAND\_LENGTH] = “/dir5/dir1/file1”;

inode\_number = open\_t(pathname);

The test cases are shown below:

|  |  |
| --- | --- |
| filepath | note |
| / | The root directory |
| /dir5 | One of the 1st level directory |
| /dir5/dir1 | One of the 2nd level directory |
| /dir5/dir1/file1 | A file under the 2nd level directory |
| /dir5/dir8/dir12/file2 | A file under the 3rd level directory |

The test code is shown blow:

#include "call.h"

int main (int argc, char \*argv[])

{

char filename[5][MAX\_COMMAND\_LENGTH] = {"/", "/dir5", "/dir5/dir1", "/dir5/dir1/file1", "/dir5/dir8/dir12/file2"};

int expected[5] = {0, 1, 5, 13, 14};

//Start testing

for(int i = 0; i < 5; i++)

{

int inode\_number = open\_t(filename[i]);

printf("======case %d: open \'%s\' =======\n", i, filename[i]);

printf("returned inode number: %d\t expected result: %d\n\n", inode\_number, expected[i]);

}

return 0;

}

1. read\_test()

Then use read\_t() reading some content of file1 to buffer and display read size. In grading process, we will use similar approach to test your code.

Here are several suggested cases:

read\_t (inode\_number, **offset**, buf, **count**)

(Only the offset and count will be changed for each case.)

|  |  |  |  |
| --- | --- | --- | --- |
| **offest** | **count** | **note** | |
| 0 | 100 | Begin at direct block, end in the same block | Access only one block |
| 4100 | 1000 | Begin at direct block, end in the same block |
| 8500 | 300 | Begin at indirect block, end in the same block |
| 40965 | 800 | Begin at indirect block, end in the same block |
| 15 | 5000 | Begin at direct block, end in another direct block | Access multiple blocks |
| 100 | 50000 | Begin at direct block, end in indirect block |
| 9000 | 60000 | Begin at indirect block, end in indirect block |
| File\_size -50 | 10000 |  | Overflowing original file test |
| File\_size + 50 | 10 |  | Offset out of range |
| 10 | MAX\_FILE\_SIZE-100 |  | Access the maximum number of blocks |

The test code is shown below:

#include "call.h"

int main (int argc, char \*argv[])

{

//argv[1]= A new file in SFS with full pathname

char filename[MAX\_COMMAND\_LENGTH]="/dir5/dir1/file1";

/\*

Allocate a buf with MAX\_FILE\_SIZE.

\*/

char buf[MAX\_FILE\_SIZE];

int read\_size;

int test\_inode=open\_t(filename);

//Start testi

int offset\_list[10] = {0, 4100, 8500, 40965, 15, 100, 9000, 1048576 - 50, 1048576 + 50, 10};

int count\_list[10] = {100, 1000, 300, 800, 5000, 50000, 60000, 10000, 10, MAX\_FILE\_SIZE - 100};

int expected[10] = {100, 1000, 300, 800, 5000, 50000, 60000, 50, 0, 1048566};

//read\_t test

for(int i = 0; i < 10; i++)

{

int cnt = count\_list[i];

int off = offset\_list[i];

printf("====case %d: read %d bytes from %d offest=======\n", i, cnt, off);

read\_size = read\_t(test\_inode, off, buf, cnt);

buf[read\_size] = '\0';

printf("read size: %d\t expected: %d\n\n",read\_size, expected[i]);

}

return 0;

}