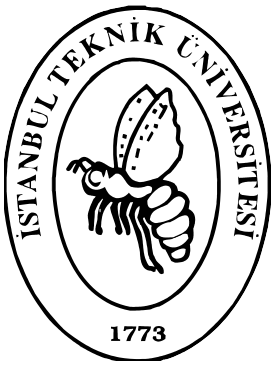
ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT



BLG-413E

SYSTEM PROGRAMMING

PROJECT 3

Ayşe Şima UYAR

EMRE UYSAL 150160510
TUGAY ACAR 150160511
CİHAT BOSTANCI 150160542

INTRODUCTION

In this project, it is expected to implement a fuse filesystem which has capability of containing website files and serves it as html files are tidied using libtidy. Only constraint is html directory must be read only.

Figure-1: Expected Structure

FIRST STEP

In the first step, we worked on libtidy. First we installed libtidy by using following commands.

```
git clone git@github.com:htacg/tidy-html5.git
cd {your-tidy-html5-directory}/build/cmake
cmake ../.. -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr
make
sudo make install
```

Figure-2: Install libtidy

After installation of libtidy we created a "new.c" file to use and learn libtidy. When we tried to compile the file we understood that "–I/usr/include/tidy –ltidy" must be written. In this file we wrote a code that takes a file content and gives the output as tidied. After reading some configuration header files we implemented our code output as giving xhtml with 4 indentation spaces.

```
ok = tidyOptSetBool( tdoc, TidyXhtmlOut, yes );  // Convert to XHTML
ind = tidyOptSetInt(tdoc,TidyIndentContent,yes); // Make indentation possible
indentation = tidyOptSetInt(tdoc,TidyIndentSpaces,4);   // set indent space to 4
```

Figure-3: Xhtml and Indentation Configuration

SECOND STEP

In this step, we tried to compile given examples in the lecture. We compiled and run them. After that we worked on them and we understood given functions in them and we made our filesystem for a try. After our basic filesystem we added read only option to our filesystem and we made all filesystem read only. So we completed our first target in Html directory. After that we added our tidy implementation and tested for different input files. In this step I must say that, we changed input part of our code because input was in buffer. So we tested this step by using debug screen.

THIRD STEP

In this step, we wrote the tidied output into buffer by using tidySaveString() function. After that we made our returned byte number output size. After that we tested our buffer in the debug screen. Also we used memset function at the start of the read function because buffer must be cleared before filled.

```
(void)fi;
memset(buf,0,size);                 // clear the last content for reading properly
```

Figure-4: Clear the buffer

```
    tidySaveString(tdoc,buf,&output.size);                  // save tidy output to buffer to show user
    res = output.size;                                      // make res equals to output size for returning byte number
    global_file_size = res;                                 // make global file size number returned byte
    printf("RESULT IS: \n %s  %d\n",buf,res);               // see the result in the buffer in debug screen
}
```

Figure-5: Write into buffer

FOURTH STEP

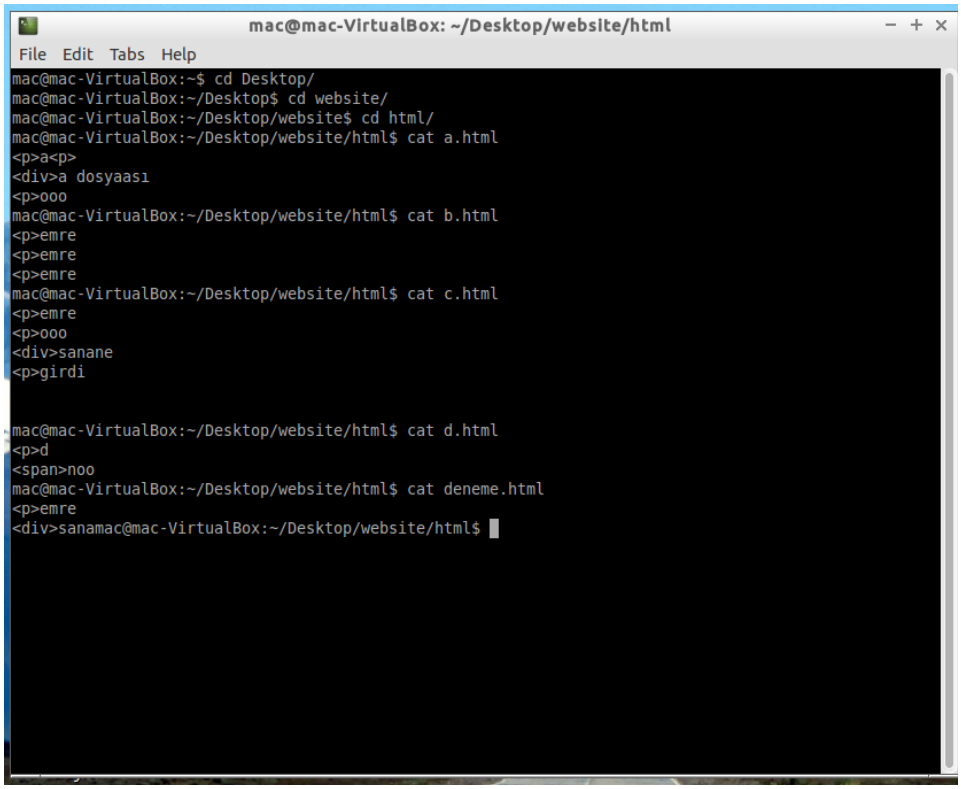
In this step, we changed getattr() function, because for a true output we need to give a true size and true file permissions to it. So, we used stat structure to specify paths attributes.

```
memset(st_data, 0, sizeof(struct stat));
// if path include .html string then it needs to be tidied
// else it no tidy opearation
if(strstr(upath,".html")==NULL)
{
    res = lstat(upath, st_data);
}
else
{
    st_data->st_mode = S_IFREG | 0444;  // it shows it is a regular file and just read
    st_data->st_nlink = 1;              // give one hardlink to point it
    st_data->st_size = global_file_size;// set the bytes for reading file
}
```

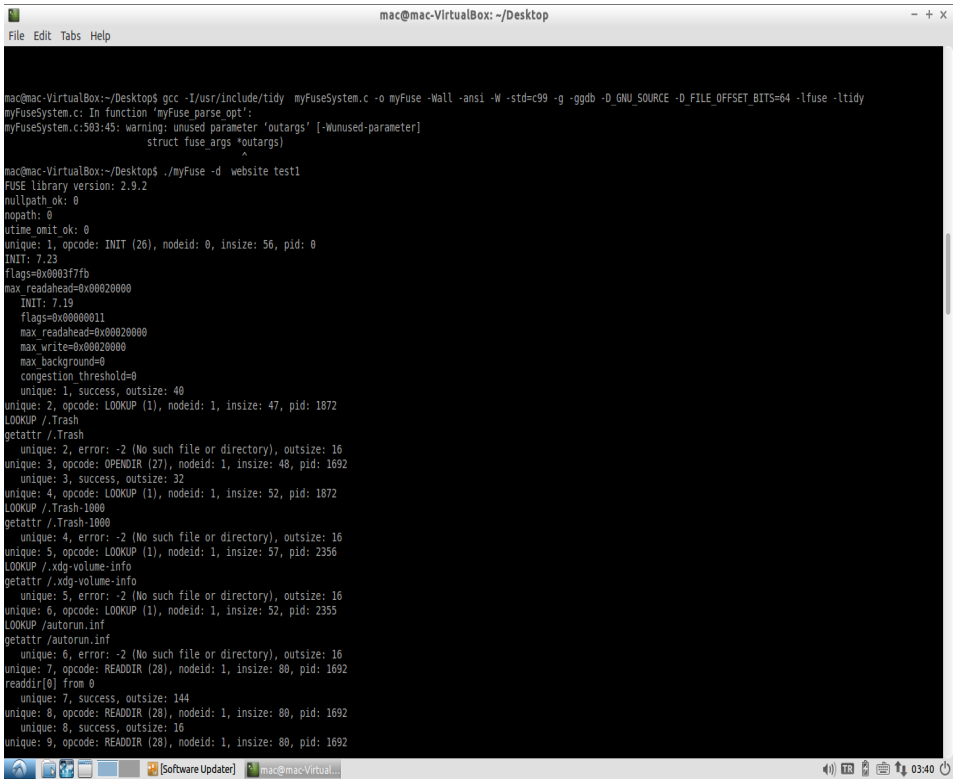
Figure-6: getattr() function

OUTPUTS

1. Picture
   This picture shows that in our website/html/ path html files are not tidied.



2. Picture
   This picture shows that how to compile and run our filesystem.



3. Picture
   Tidied output of the html files.