week4.md

Operating System Lab - CSE 341

Faculty: *Prof. Mansukh Savaliya*

Project Progress Report: *Week 4*

Group Number: *3*

Group Members:

Name	Enrolment Number
Yashil Depani	AU1841005
Vidish Joshi	AU1841019
Manav Patel	AU1841037

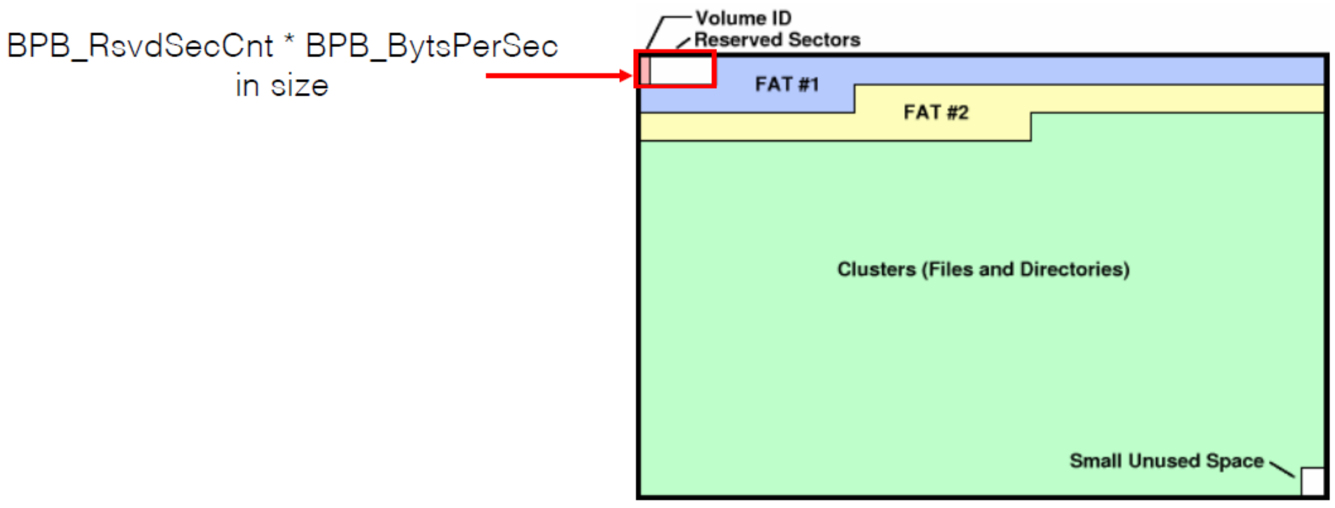
OSProject - toaruOS

An operating systems project on understanding and solving issues for the open source OS - toaruOS.

During this week, our main focus was to go head with the coding part and develop the program which allows us to open the image file and also gives us the pointers to the root directory location in the File Allocation Table to be able to operate on it.

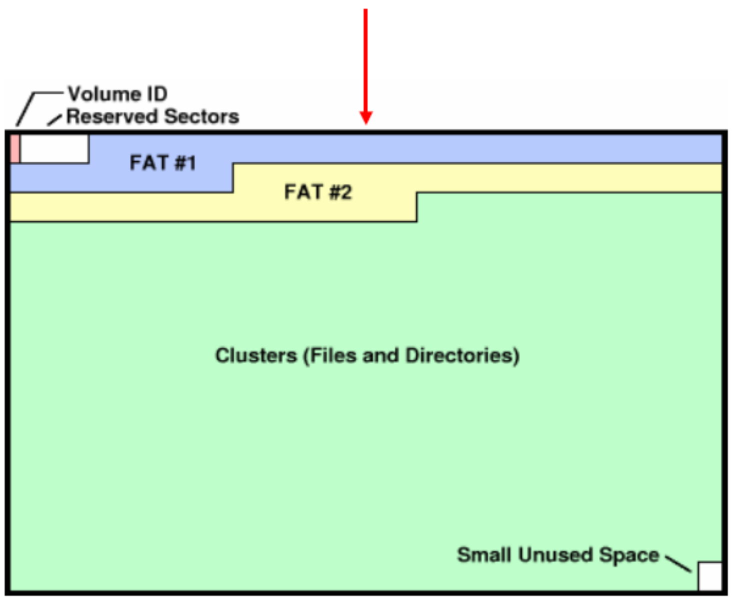
Necessary relationships between fields to access the regions of the FAT32 configured image file:

1. FAT #1 starts at address BPB_RsvdSecCnt * BPB_BytsPerSec


BPB_RsvdSecCnt * BPB_BytsPerSec in size
Volume ID
Reserved Sectors
FAT #1
FAT #2
Clusters (Files and Directories)
Small Unused Space

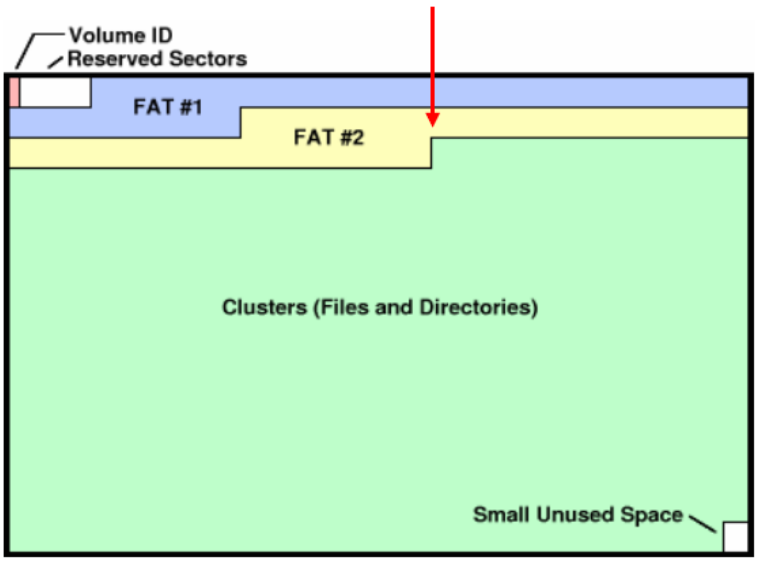
2. Each FAT has 1 32-bit word for every cluster. Each entry is the logical block of the next block in the file.

Total FAT size is BPB_NumFATs * BPB_FATSz32 * BPB_BytsPerSec



4. Clusters start at address (BPB_NumFATs * BPB_FATSz32 * BPB_BytsPerSec) + (BPB_RsvdSecCnt * BPB_BytsPerSec)

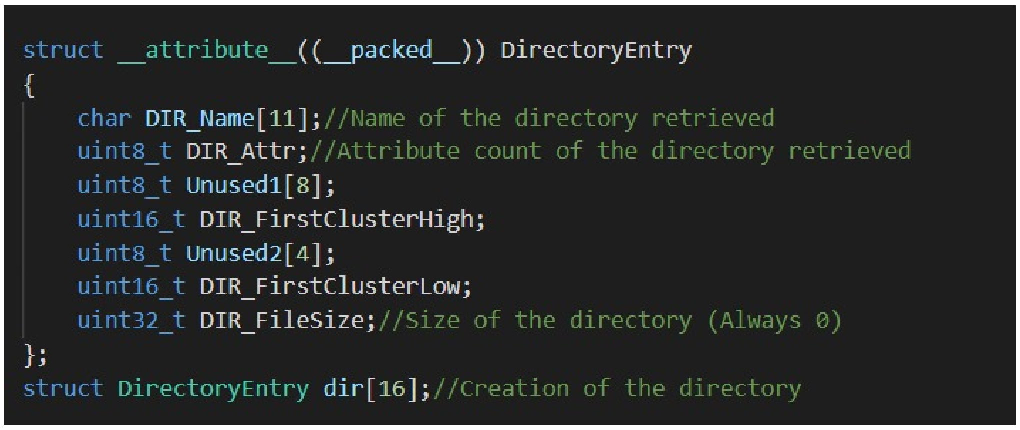
Clusters start at address (BPB_NumFATs * BPB_FATSz32 * BPB_BytsPerSec) + (BPB_RsvdSecCnt * BPB_BytsPerSec)



We use these relationships to create functions which allow us to open the `.img` file and store all the fields and necessary offsets with their appropriate size in variables.

Created Functions:

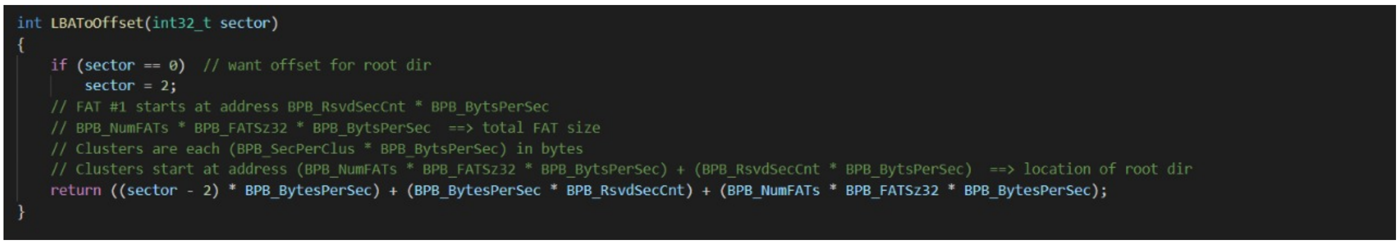
This the structure of a directory. We store all the information of a *dir* in a variable of this structure to access the its properties such as FileSize, DIrName, FirstCluster properties, etc. as shown in the image below,



```c
struct __attribute__((__packed__)) DirectoryEntry
{
    char DIR_Name[11];//Name of the directory retrieved
    uint8_t DIR_Attr;//Attribute count of the directory retrieved
    uint8_t Unused1[8];
    uint16_t DIR_FirstClusterHigh;
    uint8_t Unused2[4];
    uint16_t DIR_FirstClusterLow;
    uint32_t DIR_FileSize;//Size of the directory (Always 0)
};
struct DirectoryEntry dir[16];//Creation of the directory
```

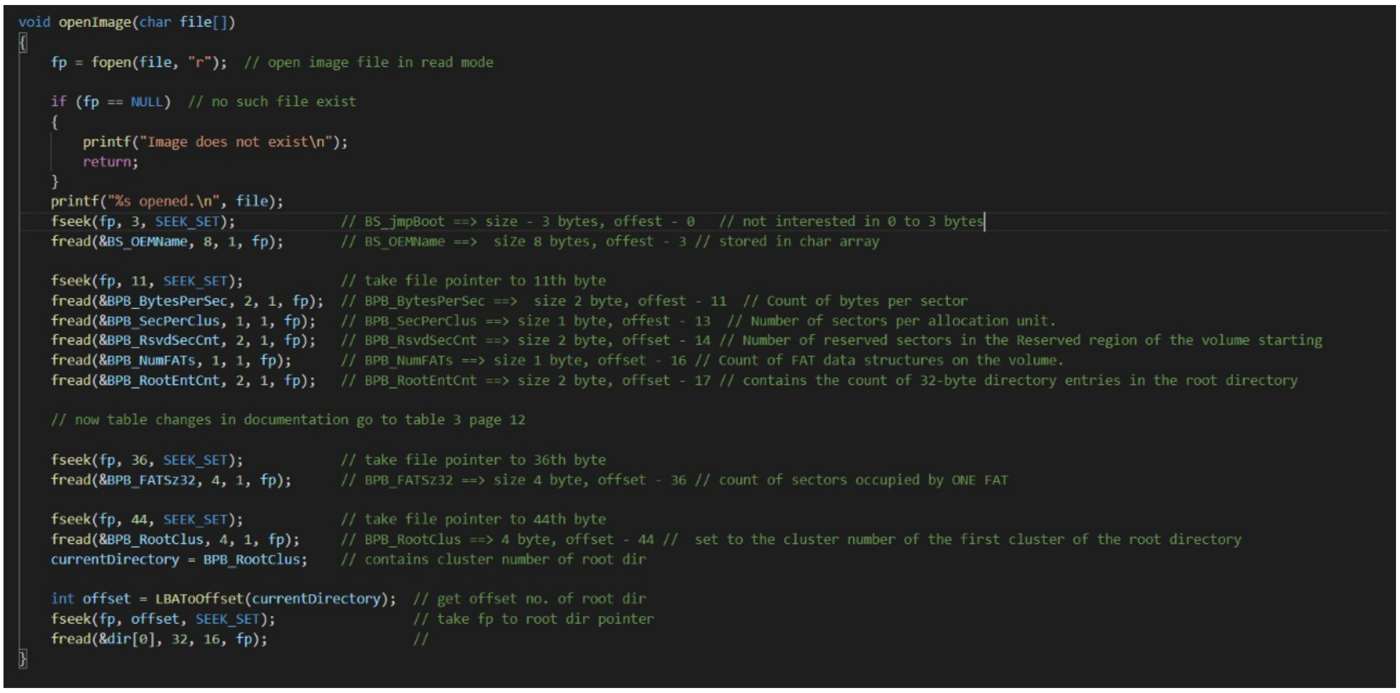
When we retrieve the location of directories from the Allocation Table, the location address follows *Logical Block Addressing* method which is useful for the machine to locate directories. We convert it into readable form of memory offset.

This conversion can be understood and achieved from the relationships shown in the first section and the *documentation*.



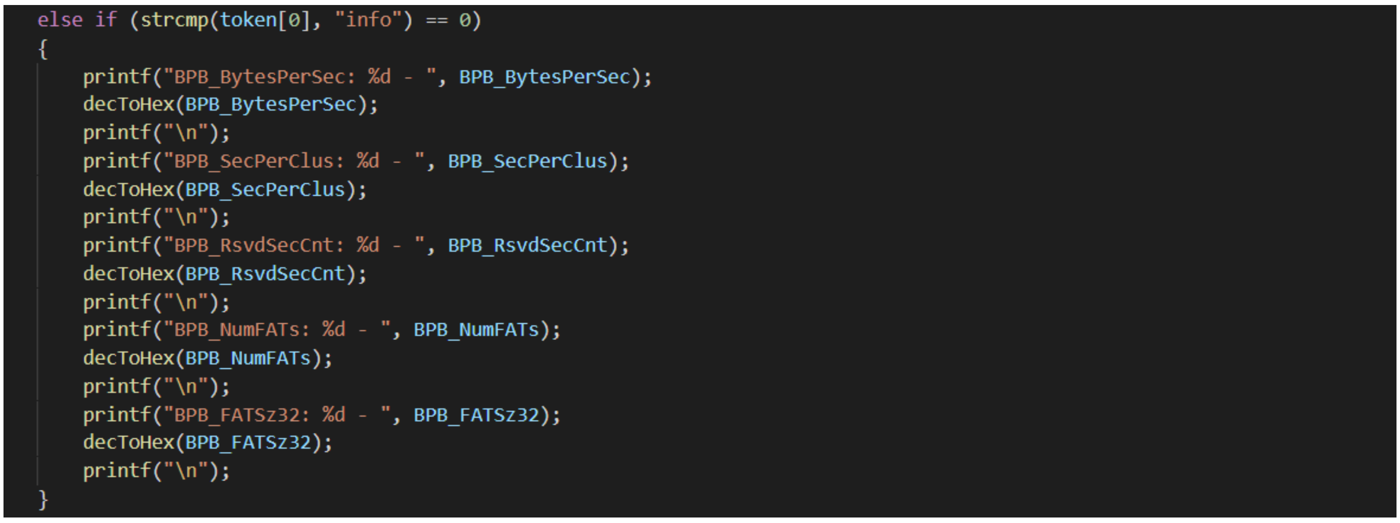
```c
int LBAToOffset(int32_t sector)
{
    if (sector == 0)  // want offset for root dir
        sector = 2;
    // FAT #1 starts at address BPB_RsvdSecCnt * BPB_BytsPerSec
    // BPB_NumFATs * BPB_FATSz32 * BPB_BytsPerSec  ==> total FAT size
    // Clusters are each (BPB_SecPerClus * BPB_BytsPerSec) in bytes
    // Clusters start at address (BPB_NumFATs * BPB_FATSz32 * BPB_BytsPerSec) + (BPB_RsvdSecCnt * BPB_BytsPerSec)  ==> location of root dir
    return ((sector - 2) * BPB_BytesPerSec) + (BPB_BytesPerSec * BPB_RsvdSecCnt) + (BPB_NumFATs * BPB_FATSz32 * BPB_BytesPerSec);
}
```

This is the `openImage` function that allows us to open the `.img` file containing the FAT configured drive. As described in the previous weekly reports regarding the offsets in the Allocation table, their sizes and their importance, we store these necessary values after opening the file in appropriate data structures using *file pointer* and related functions such `fseek` and `fread` .



```c
void openImage(char file[])
{
    fp = fopen(file, "r");  // open image file in read mode

    if (fp == NULL)  // no such file exist
    {
        printf("Image does not exist\n");
        return;
    }
    printf("%s opened.\n", file);
    fseek(fp, 3, SEEK_SET);             // BS_jmpBoot ==> size - 3 bytes, offest - 0   // not interested in 0 to 3 bytes
    fread(&BS_OEMName, 8, 1, fp);       // BS_OEMName ==>  size 8 bytes, offest - 3 // stored in char array

    fseek(fp, 11, SEEK_SET);            // take file pointer to 11th byte
    fread(&BPB_BytesPerSec, 2, 1, fp);  // BPB_BytesPerSec ==>  size 2 byte, offest - 11  // Count of bytes per sector
    fread(&BPB_SecPerClus, 1, 1, fp);   // BPB_SecPerClus ==> size 1 byte, offest - 13  // Number of sectors per allocation unit.
    fread(&BPB_RsvdSecCnt, 2, 1, fp);   // BPB_RsvdSecCnt ==> size 2 byte, offset - 14 // Number of reserved sectors in the Reserved region of the volume starting
    fread(&BPB_NumFATs, 1, 1, fp);      // BPB_NumFATs ==> size 1 byte, offset - 16 // Count of FAT data structures on the volume.
    fread(&BPB_RootEntCnt, 2, 1, fp);   // BPB_RootEntCnt ==> size 2 byte, offset - 17 // contains the count of 32-byte directory entries in the root directory

    // now table changes in documentation go to table 3 page 12

    fseek(fp, 36, SEEK_SET);            // take file pointer to 36th byte
    fread(&BPB_FATSz32, 4, 1, fp);      // BPB_FATSz32 ==> size 4 byte, offset - 36 // count of sectors occupied by ONE FAT

    fseek(fp, 44, SEEK_SET);            // take file pointer to 44th byte
    fread(&BPB_RootClus, 4, 1, fp);     // BPB_RootClus ==> 4 byte, offset - 44 //  set to the cluster number of the first cluster of the root directory
    currentDirectory = BPB_RootClus;    // contains cluster number of root dir

    int offset = LBAToOffset(currentDirectory);  // get offset no. of root dir
    fseek(fp, offset, SEEK_SET);                 // take fp to root dir pointer
    fread(&dir[0], 32, 16, fp);                  //
}
```

This function is used to implement the `info` command on an already opened `.img` file. This function prints all the important fields as mentioned in the documentation that we stored in variable after opening the image file.



```
else if (strcmp(token[0], "info") == 0)
{
    printf("BPB_BytesPerSec: %d - ", BPB_BytesPerSec);
    decToHex(BPB_BytesPerSec);
    printf("\n");
    printf("BPB_SecPerClus: %d - ", BPB_SecPerClus);
    decToHex(BPB_SecPerClus);
    printf("\n");
    printf("BPB_RsvdSecCnt: %d - ", BPB_RsvdSecCnt);
    decToHex(BPB_RsvdSecCnt);
    printf("\n");
    printf("BPB_NumFATs: %d - ", BPB_NumFATs);
    decToHex(BPB_NumFATs);
    printf("\n");
    printf("BPB_FATSz32: %d - ", BPB_FATSz32);
    decToHex(BPB_FATSz32);
    printf("\n");
}
```

This code is maintained in this GitHub repository.

Plan for next week

To summarize, with the help of the above code, we are able to open a *FAT32* configured image file and access the directory structure stored in the File Allocation Table inside it. Our next aim is to open, read and print this directory structure with its directories, sub-directories and files.

Contribution

Writing and implementing above code and functionalities - All 3 members