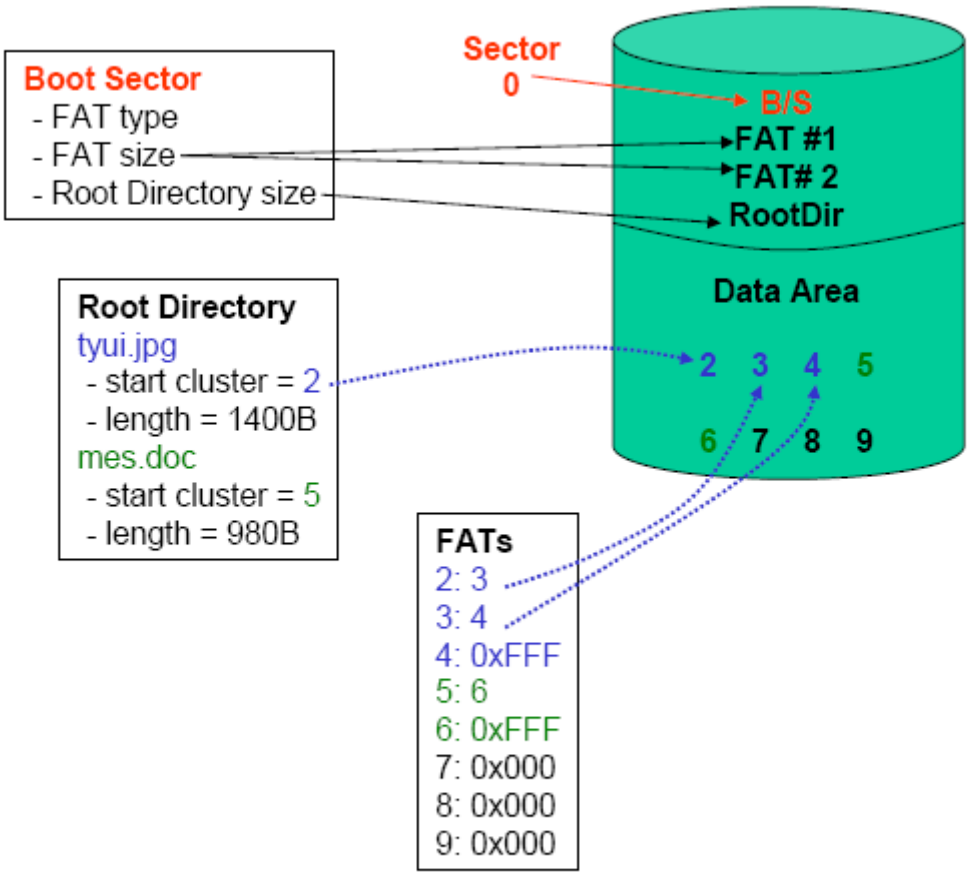
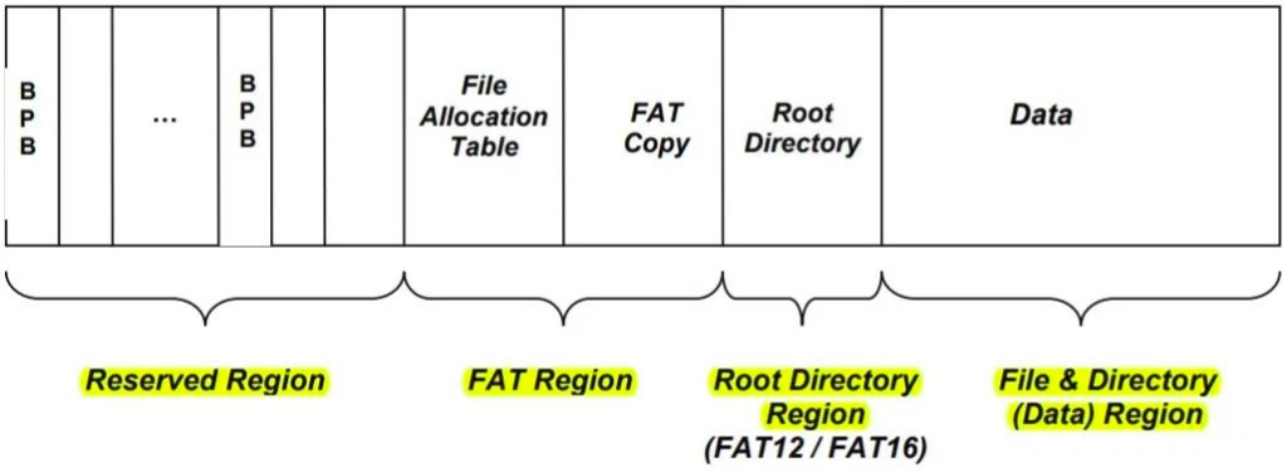
CSE-312 OPERATİNG SYTEM HW4-REPORT

REFİK ORKUN ARSLAN 151044063



The name of the file system originates from the file system's prominent usage of an index table, the File Allocation table, statically allocated at the time of formatting. The table contains entries for each *cluster*, a contiguous area of disk storage. Each entry contains either the number of the next cluster in the file, or else a marker indicating end of file, unused disk space, or special reserved areas of the disk. The *root directory* of the disk contains the number of the first cluster of each file in that directory; the operating system can then traverse the FAT table, looking up the cluster number of each successive part of the disk file as a *cluster chain* until the end of the file is reached. In much the same way, *sub-directories* are implemented as special files containing the *directory entries* of their respective files. **We did the navigation between the classes with the cluster chain.**



FAT: Reserved Area

The *first section* of the FAT file system is the reserved area. *Reserved area includes data* in the file system category. It *usually* occupies 1 sector in FAT12 and FAT16, the exact size is defined in the Boot sector, which is also located here in the 1st sector of the volume.

FAT: FAT Area

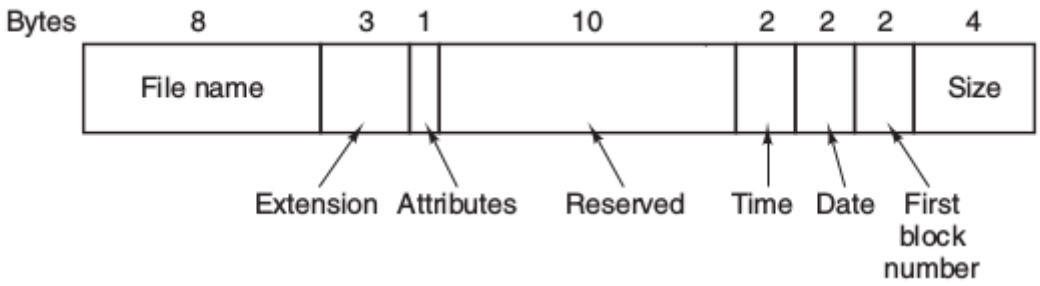
FAT Area is used to store the primary and backup FAT structures. This section starts in the next following sector, straight after the reserved area section. *FAT Area's size depends on the size and number of FAT structures*.

FAT: Root Directory Area

Root Directory Area exists only on FAT12/FAT16 file systems.

FAT: Data Area

Data Area is the 3rd section, which contains the clusters that will be allocated for storing the files and directories contents.



It shows the contents of dictionaries found in this data area.

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

We will calculate according to FAT 12, the block will give you 4096 times the total memory.

Format function:

First create a blok.Overwrite whole disk with clean block (necessary to avoid any clutter).Write first block.After set FAT table structure.Create a new disk block for root.Set directory and traversal between block.

FAT[0] = ENDOFCHAIN;
FAT[1] = 2;
FAT[2] = ENDOFCHAIN;
FAT[3] = ENDOFCHAIN;

By connecting the FATs, it is navigated between the blocks and the appropriate place is searched.

Create a block for to store the root directory
update the location of the root dir

diskblock_t virtual_disk[MAXBLOCKS]; define our in-memory virtual, with MAXBLOCKS blocks

fatentry_t FAT[MAXBLOCKS]; define a file allocation table with MAXBLOCKS 16-bit entries

fatentry_t root_dir_index = 0; rootDir will be set by format

direntry_t *current_dir = NULL; use this to track the location of the first block in a dir block chain

fatentry_t current_dir_index = 0;

MKDİR

Implements the creation of a directory for a given path

```
void mymkdir(char *path) {
  int initial_current_dir_index = current_dir_index;
  int initial_current_dir_first_block = current_dir->first_block;

  // if the path is an absolute path switch to root before creating
  // CONFIRM IF THIS IS REQUIRED
  if (path[0] == '/') {if the path is an absolute path switch to root before creating
    current_dir_index = root_dir_index;
    current_dir->first_block = root_dir_index;
  }

  // not sure why I can't just use path here, life is too short
  char str[strlen(path)];
  strcpy(str, path);

  char *dir_name = strtok(str, "/");

  while (dir_name) {
    allocate a new block for the subdir
      int sub_dir_block_index = next_unallocated_block();
      create_block(sub_dir_block_index, DIR);


      virtual_disk[sub_dir_block_index].dir.entrylist[0].unused = FALSE;
      virtual_disk[sub_dir_block_index].dir.entrylist[0].first_block = sub_dir_block_index;
      strcpy(virtual_disk[sub_dir_block_index].dir.entrylist[0].name, "..");


      add_block_to_directory(sub_dir_block_index, dir_name, TRUE);add to the parents entry list


      current_dir_index = sub_dir_block_index;'cd' to the new sub dir
      current_dir->first_block = sub_dir_block_index;

      //set the name for the next level
      dir_name = strtok(NULL, "/");
  }
```

```
 /move back to the original dir index
 current_dir_index = initial_current_dir_index;
 current_dir->first_block = initial_current_dir_first_block;
}
```

RMDİR

```
void myrmdir(char *path){
 if it's an abs path then return to root
 if (path[0] == '/'){
  mychdir("root");
 }

  if the folder is empty
 if (strcmp(mylistdir(path)[1], "ENDOFDIR") == 0) {
  // keep a track of where we start the process as the current dir changes in here, somewhere...
  int initial_current_dir_index = current_dir_index;
  int initial_current_dir_first_block = current_dir->first_block;

  calculate the parent of the dir to remove and chdir to it
  char *target = strstr(path, last_entry_in_path(path_to_array(path)));
  int position = target - path;
  char parent_path[position+1];
  for(int i = 0; i < position + 1; i++){
   parent_path[i] = '\0';
  }

  strncpy(parent_path, path, position);
  if(strcmp(parent_path,"") != 0) {
   mychdir(parent_path);
  }
  find the entry for the dir to be deleted
  int entrylist_target = file_entry_index(last_entry_in_path(path_to_array(path)));

  find the start of it's block chain
  int block_chain_target = virtual_disk[current_dir_index].dir.entrylist[entrylist_target].first_block;

  // clear the dir entry and it's name ready for reuse (still don't have that feature)
  direntry_t *dir_entry = &virtual_disk[current_dir_index].dir.entrylist[entrylist_target];
  dir_entry->first_block = 0;
  dir_entry->unused = 1;
  int length = strlen(dir_entry->name);
  for (int i = 0; i < length; i++){
   dir_entry->name[i] = '\0';
  }

  /clear the dirs block chain and update the fat
  int next_block_chain_target;
  while(1){
   next_block_chain_target = FAT[block_chain_target];
   FAT[block_chain_target] = UNUSED;
   if (next_block_chain_target == ENDOFCHAIN) break;
   block_chain_target = next_block_chain_target;
  }
  copy_fat(FAT);


  current_dir_index = initial_current_dir_index;
  current_dir->first_block = initial_current_dir_first_block;
 } else {
  printf("That directory has content and cannot be deleted.\n");
 }
}
```

WRİTE

```
void write_block(diskblock_t *block, int block_address, char type)
{
 if (type == 'd') { //block is data
  memmove(virtual_disk[block_address].data, block->data, BLOCKSIZE);
 }
 else if (type == 'f') { // block is fat
```

```
    memmove(virtual_disk[block_address].fat, block->fat, BLOCKSIZE);
  }
  else if (type == 'r') { //block is dir, CHECK THIS, dir not working
    memmove(virtual_disk[block_address].data, block->data, BLOCKSIZE);
  }
  else {
    printf("Invalid Type");
  }
}
```

I created the dictionary of orkun, then I created the refik sub-dictionary, which is its sub-dictionary, and the arslan dictionary with its sub-dictionary was created, and then this dictionary was deleted. The dictionary was also printed on the screen inside. Although it was executed every time in the terminal, the dictionaries were kept as they were written to disk.

```
int next_unallocated_block()
{
  for(int i = 0; i < MAXBLOCKS; i++){
    if (FAT[i] == UNUSED){
      FAT[i] = ENDOFCHAIN;
      copy_fat(FAT);
      return i;
    }
  }
  return -1;
}
```

Traversal blocks next block allocation