File System Management

COP-5614 Operating System
Group 9:

Md Shahadat Iqbal Md Abdullah Al Mamun Vitalii Stebliankin

Contributions

Name	Contribution
Md Abdullah Al Mamun	**Bitmap and File name checking** • `bitmap_init()` • `bitmap_first_unused()` • `bitmap_reset()` • `illegal_file_name()`

Contributions

Name	Contribution
Md Abdullah Al Mamun	**Bitmap and File name checking** • `bitmap_init()` • `bitmap_first_unused()` • `bitmap_reset()` • `illegal_file_name()`
Vitalii Stebliankin	**Remove** • `File Unlink()` • `Dir Unlink()` • `remove_file_or_directory()` • `remove_inode()`

Contributions

Name	Contribution
Md Abdullah Al Mamun	**Bitmap and File name checking** • `bitmap_init()` • `bitmap_first_unused()` • `bitmap_reset()` • `illegal_file_name()`
Vitalii Stebliankin	**Remove** • `File Unlink()` • `Dir Unlink()` • `remove_file_or_directory()` • `remove_inode()`
Md Shahadat Iqbal	**Read/Write** • `File_Read()` • `File_Write()` • `File_Seek()` • `Dir_Read()` • `Dir_Size()`

Bitmap Initialization

- **Set all ‘0’**
  - memset()
- **Set first nbits ‘1’**

Buffer = 0

1 (OR)

---

1 (*nbits*)

- **Features**
  - If *nbits* is zero
  - No further checking

```
static void bitmap_init(int start, int num, int nbits){
  int bit, byte, sector;

  //initializing 0 in bitmap of every sector
  int flag1 = 1;
  sector = 0;
  while(sector < num){
    char buffer_bitmap[SECTOR_SIZE];            // bitma
    memset(buffer_bitmap, 0, SECTOR_SIZE);      // fill
    for (byte = 0; byte < SECTOR_SIZE; byte++){ // loopi
      if(!flag1) {
        break;
      }
      else {
        for (bit = 7; bit >= 0; bit--){           // loo
        if (nbits-- > 0){                         // checc
          buffer_bitmap[byte] |= (1 << bit);
          }
        else{
          flag1 = 0;
          break;
          }             // set 1 for first nbits (bitwis
        }
      }
    }
    //update on the disk
    Disk_Write(start+sector, buffer_bitmap);
    sector++;
  }
}
```

Find First Unused

- **Bitwise AND**

Buffer= 0

1 (AND)

0

- **Flip “0” to “1”, return loc**

Buffer = 0

1 (OR)

1

- **Features**
  - If id > nbits
  - No further checking

```
static int bitmap_first_unused(int start, int num, int nbits){
  int id = 0; //starting index is zero by default
  char check_bit_empty;
  char buffer_bitmap[SECTOR_SIZE];
  int bit, byte, sector;

  // checking every sectors to find the first unused memory
  sector = 0;
  while(sector < num){                          // looping eacl
    Disk_Read(start+sector, buffer_bitmap);     // pull curren
    for (byte = 0; byte < SECTOR_SIZE; byte++){ // looping eacl
      for (bit = 7; bit >= 0; bit--){           // looping eve
        check_bit_empty = (buffer_bitmap[byte] >> bit) & 1; //

        if (check_bit_empty == 0){              // if used bit
          buffer_bitmap[byte] |= 1 << bit;
          Disk_Write(start+sector, buffer_bitmap); // update t
          return id;                            // return i
        }
        id++;
        if (id > nbits){        // return -1 if the index exce
          return -1;
        }                       // increament index by 1
      }
    }
    sector++;
  }
  return -1;                                    // return -1 i
}
```

Bitmap Reset

- **Pulling current bitmap**
- **Set ibit to “0”**
  - Bitwise AND

```
static int bitmap_reset(int start, int num, int ibit) {
  char buffer_bitmap[SECTOR_SIZE];
  int bit, byte, sector;

  // loop through each sector in bitmap
  sector = 0;
   while(sector < num){
    Disk_Read(start+sector, buffer_bitmap);
    for (byte = 0; byte < SECTOR_SIZE; byte++){  // che
      for(bit = 7; bit >= 0; bit--){             // che
        if(!ibit){                               // onc
          buffer_bitmap[byte] &= ~(1 << bit);    // Lef
          Disk_Write(start+sector, buffer_bitmap);
          return 0;
        }
        ibit--;
      }
    }
    sector++;
  }
  return -1;
}
```

Illegal filename

- **Valid characters**
  - letters
  - dots, dashes, and underscore
  - numbers

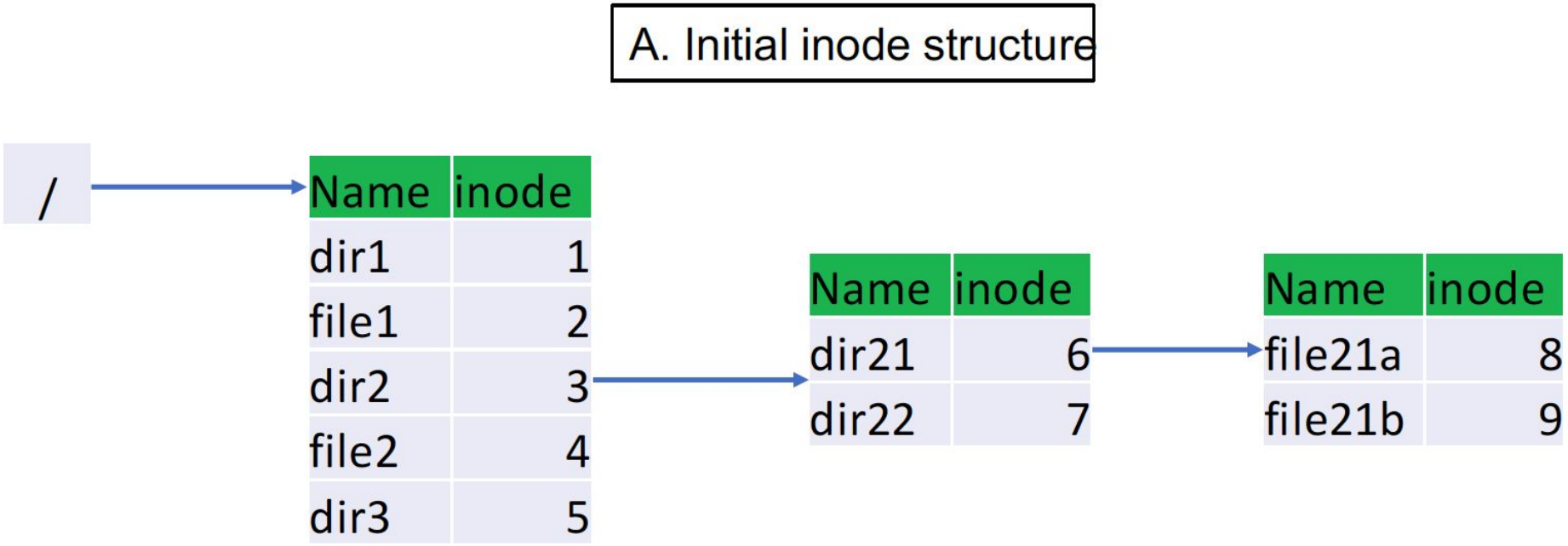
- **Length**
  - MAX_NAME-1

```
static int illegal_filename(char* name) {
  int i;
  char end[] = "-_.";

  //checking character by character
  for(i = 0; i < strlen(name); i++){
    if(!isdigit(name[i]) && !isalpha(name[i]) && !strchr(end, name[i])){
      printf("Illegal character found! %c\n", name[i]);
      return 1;
    }
  }

  //checking length
  if(strlen(name) >= MAX_NAME-1){
    printf("Bad Length\n");
    return 1;
  }

  //return false otherwise
  return 0;
}
```

Demo: Bitmap

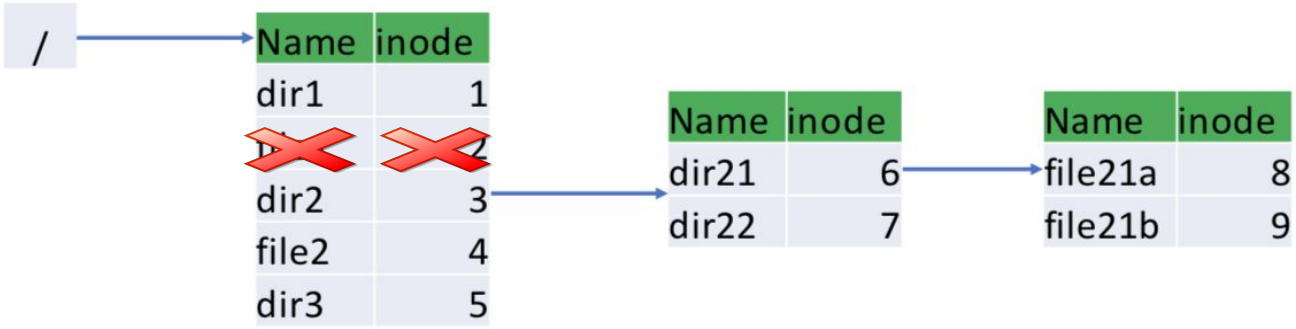
1. Simple test
   a. Create file/directory
   b. Remove or unlink file/directory

2. Customized test: **5 directories and 4 files**

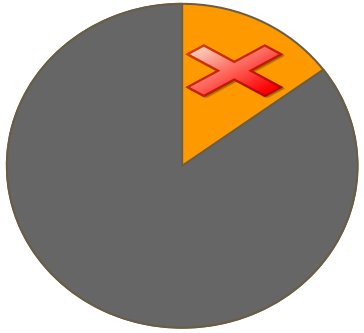


Remove File or Directory

- Remove iNode from the disk and update parent iNode



- Remove the content of file from the disk



- Update bitmaps

iNode bitmap

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	1	1	1	1	1	1	1	0	0	0	0	0

0

Disk Sector bitmap

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	1	1	1	1	1	1	1	0	1	1	0	

0

remove_inode() - Part 1

- Load iNode from the disk
- Check for errors
- Remove content from the disk

```
int remove_inode(int type, int parent_inode, int child_inode)
{
  // This function is a modification to add_inode() that removes child inode
  dprintf("... removing inode %d\n", child_inode);
  //------------------------------------------------------------------------------
  // Load Child iNode from the disk
  //------------------------------------------------------------------------------
  // load the disk sector containing the child inode
  int inode_sector = INODE_TABLE_START_SECTOR+child_inode/INODES_PER_SECTOR;
  char inode_buffer[SECTOR_SIZE];
  if(Disk_Read(inode_sector, inode_buffer) < 0){
    dprintf("Error: can't read inode from disk sector");
    return -1;
  }
  //dprintf("%s\n", inode_buffer);
  dprintf("... load inode table for child inode from disk sector %d\n", inode_sector);

  // get the child inode
  int inode_start_entry = (inode_sector-INODE_TABLE_START_SECTOR)*INODES_PER_SECTOR;
  int offset = child_inode-inode_start_entry;
  assert(0 <= offset && offset < INODES_PER_SECTOR);
  inode_t* child = (inode_t*)(inode_buffer+offset*sizeof(inode_t));
  //
  // Check for errors
  //------------------------------------------------------------------------------
  // Check if the type is right
  if(child->type != type){
    dprintf("Error: the actual type of inode doesn't match with parameter specified in function remove_inode()");
    return -3;
  }

  //Check if directory is not empty
  if(child->type==1 && child->size!=0){
    dprintf("Error: directory is not empty");
    osErrno = E_DIR_NOT_EMPTY;
    return -2;
  }
  //------------------------------------------------------------------------------
  // Remove the file content from the disk
  //------------------------------------------------------------------------------
  char buf_sector[SECTOR_SIZE];
  int sector_address;
  if(child->size>0){
    dprintf("%s\n", "... Removing file data from the disk");
    for(int i; i<MAX_SECTORS_PER_FILE; i++){
      sector_address = child->data[i];
      if(sector_address>0){
        //remove from sector
        if(Disk_Read(sector_address, buf_sector)<0) return -1; //try to read the sector
        memset(buf_sector, 0, SECTOR_SIZE);//set buffer to 0
        Disk_Write(sector_address, buf_sector); //write 0s to the sector
        bitmap_reset(SECTOR_BITMAP_START_SECTOR, SECTOR_BITMAP_SECTORS, sector_address);//reset bitmap for the current sector
      }
    }
  } else{
    dprintf("%s\n", "... The directory/file is empty");
  }
  //------------------------------------------------------------------------------
```

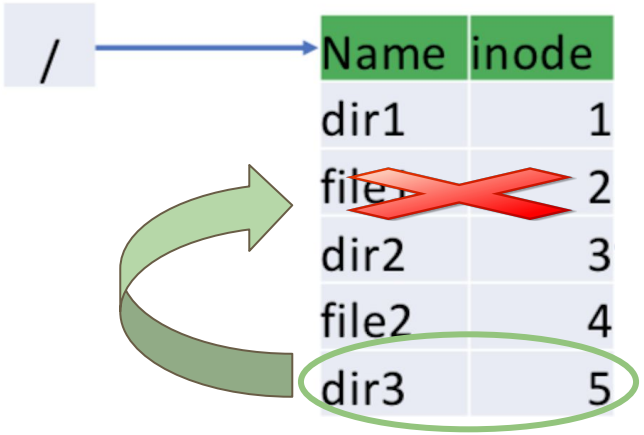
remove_inode() - Part 2

- Remove iNode from the disk
- Reset iNode bitmap
- Read parent iNode to the disk

```
301   //------------------------------------------------------------------------------
302   // Remove iNode from the disk sector
303   //------------------------------------------------------------------------------
304   dprintf("%s\n", "... Removing inode from the disk");
305   memset(child, 0, sizeof(inode_t)); //Delete child inodes from memory
306  // memset(inode_buffer, 0, SECTOR_SIZE);
307   if(Disk_Write(inode_sector, inode_buffer)<0) return -1;
308   //------------------------------------------------------------------------------
309   // Reset iNode bitmap
310   //
311   bitmap_reset(INODE_BITMAP_START_SECTOR, INODE_BITMAP_SECTORS, child_inode); //reset i-node bitmap
312   // Remove link from parent inode to child inode
313   //------------------------------------------------------------------------------
314   // Read the parent iNode from the disk
315   //------------------------------------------------------------------------------
316   // get the disk sector containing the parent inode
317   inode_sector = INODE_TABLE_START_SECTOR+parent_inode/INODES_PER_SECTOR;
318   if(Disk_Read(inode_sector, inode_buffer) < 0) return -1;
319   dprintf("... load inode table for parent inode %d from disk sector %d\n",
320    parent_inode, inode_sector);
321
322   // get the parent inode
323   inode_start_entry = (inode_sector-INODE_TABLE_START_SECTOR)*INODES_PER_SECTOR;
324   offset = parent_inode-inode_start_entry;
325   assert(0 <= offset && offset < INODES_PER_SECTOR);
326   inode_t* parent = (inode_t*)(inode_buffer+offset*sizeof(inode_t));
327   dprintf("... get parent inode %d (size=%d, type=%d)\n",
328    parent_inode, parent->size, parent->type);
329   //------------------------------------------------------------------------------
330   // Check if parent iNode is a directory
331   //------------------------------------------------------------------------------
332   if(parent->type != 1){
333     dprintf("... error: parent inode is not directory\n");
334     return -2;
335   }
```

remove_inode() - Part 3

- Update parent iNode



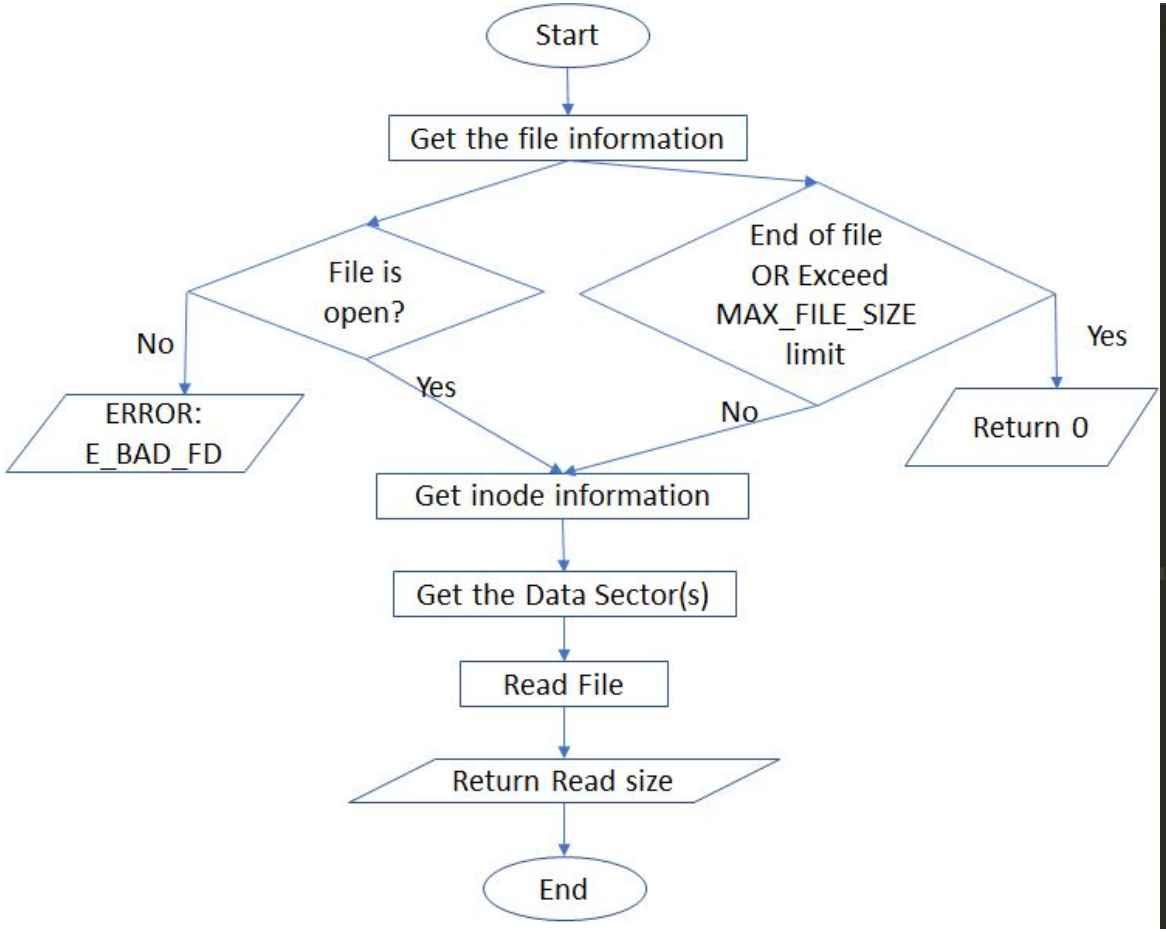

```
336   //------------------------------------------------------------------------------------------------
337   // Update parent iNode and dirent table
338   //------------------------------------------------------------------------------------------------
339   int nentries = parent->size; // remaining number of directory entries
340   int idx = 0;
341   int found_flag=0;
342   while(nentries > 0) {
343     char dirent_buffer[SECTOR_SIZE]; // cached content of directory entries
344     if(Disk_Read(parent->data[idx], dirent_buffer) < 0) return -2;
345     for(int i=0; i<DIRENTS_PER_SECTOR; i++) {
346     if(i>nentries) break;
347     dirent_t* tmp_dirent = (dirent_t*)(dirent_buffer +i*sizeof(dirent_t));
348
349     if(tmp_dirent->inode==child_inode){//dirent found
350           dprintf("... Corresponding dirent found\n");
351           char dirent_buffer2[SECTOR_SIZE];
352
353           int group = parent->size/DIRENTS_PER_SECTOR;
354           if(Disk_Read(parent->data[group], dirent_buffer2) < 0) return -2;
355           int start_entry = group*DIRENTS_PER_SECTOR;
356           offset = parent->size-start_entry;
357           dirent_t* tmp_dirent1 = (dirent_t*)(dirent_buffer2+offset*sizeof(dirent_t));
358
359           strncpy(tmp_dirent->fname, tmp_dirent1->fname, MAX_NAME);
360           tmp_dirent->inode = tmp_dirent1->inode;
361           memset(tmp_dirent1, 0, sizeof(dirent_t));
362           parent->size--;
363           if(Disk_Write(inode_sector, inode_buffer)<0) return -1;//update parent inode
364           if(Disk_Write(parent->data[group], dirent_buffer2)) return -1;//update dirent
365           if(Disk_Write(parent->data[idx], dirent_buffer)) return -1;//update dirent
366           found_flag=1;
367     }
368     }
369     idx++; nentries -= DIRENTS_PER_SECTOR;
370   }
371   if(found_flag==1) return 0;
372   else{
373   dprintf("... could not find dirent\n");
374   return -1; // not found
375   }
376 }
```

Read/Write

1. File_Read()
2. File_Write()
3. File_Seek()
4. Dir_Read()
5. Dir_Size()

File Read

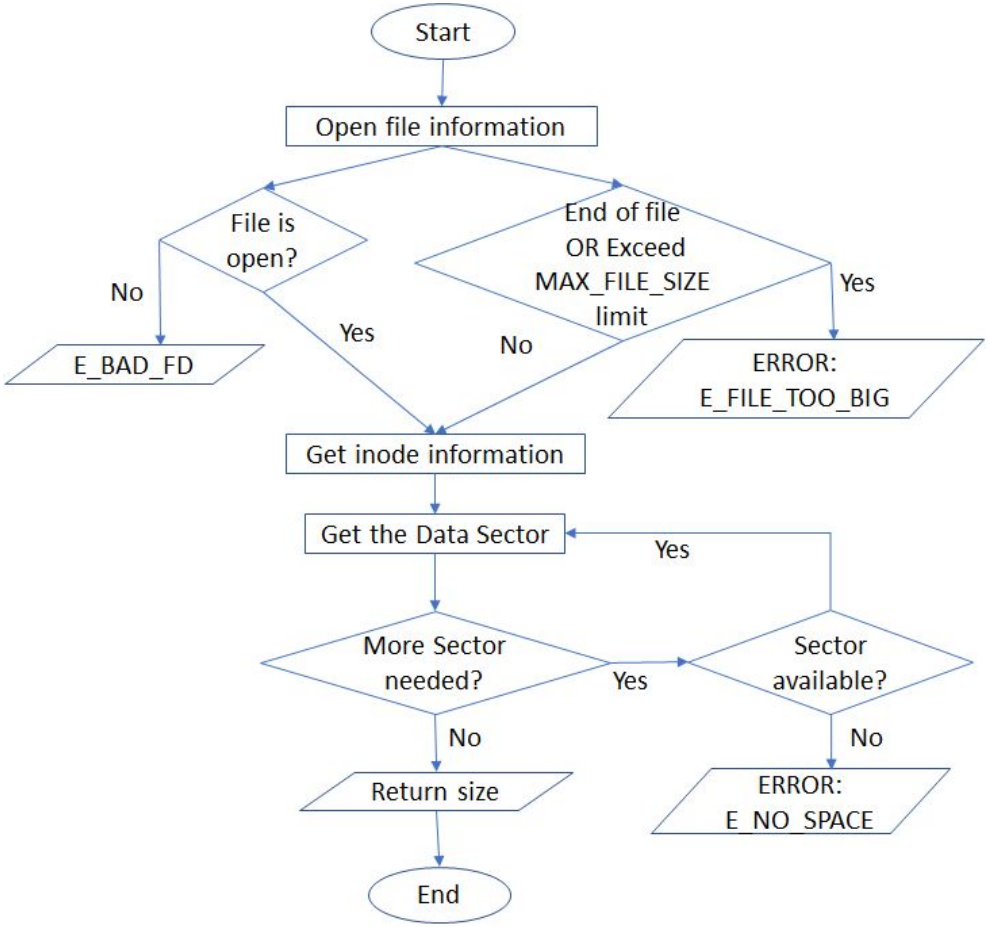
```
int File_Read(int fd, void* buffer, int size)
```




```
int File_Read(int fd, void* buffer, int size)
{
  int reading_part = 0; // File is read and kept a copy into reading_part
  open_file_t openFileInfo = open_files[fd]; // Information about the file fd
  // First check the file is open or not
  if (openFileInfo.inode == 0)
  {
      dprintf("...file not open");
      osErrno = E_BAD_FD;
      return -1;
  }
  // load the disk sector containing the inode
  int inode_sector = INODE_TABLE_START_SECTOR+ openFileInfo.inode/INODES_PER_SECTOR;
  char inode_buffer[SECTOR_SIZE];
  if(Disk_Read(inode_sector, inode_buffer) < 0) {
    osErrno = E_GENERAL;
    return -1;
  }
  dprintf("... load inode table for inode from disk sector %d\n", inode_sector);

  // get the inode
  int inode_start_entry = (inode_sector - INODE_TABLE_START_SECTOR) * INODES_PER_SECTOR;
  int offset = openFileInfo.inode - inode_start_entry;
  assert(0 <= offset && offset < INODES_PER_SECTOR);
  inode_t *child_inode = (inode_t *) (inode_buffer + offset * sizeof(inode_t));
  int sector_read = openFileInfo.pos / SECTOR_SIZE; // The sector which need to read

  // Check if the position is at the end of file, or data at that sector does not exists
  if (openFileInfo.pos == MAX_FILE_SIZE || !child_inode->data[sector_read]) {
    return 0;
  }

  char sectorBuffer[SECTOR_SIZE];
  // check if the there any problem to read the data
  if (Disk_Read(child_inode->data[sector_read], sectorBuffer) < 0) {
    return -1;
  }
  dprintf("... load disk sector %d\n", child_inode->data[sector_read]);
  void *reading_pos = sectorBuffer + (open_files[fd].pos - (openFileInfo.pos / SECTOR_SIZE) * SECTOR_SIZE);

  // Read the data from the file unlit maximum file is already read completely or
  // requeased read size is already read or all the data has been accessed
  while (open_files[fd].pos < MAX_FILE_SIZE && size > 0 && child_inode->data[sector_read]) {

      if (size >= SECTOR_SIZE) {
          dprintf("... Reading BYTE  %d\n", open_files[fd].pos);
          memcpy(buffer, reading_pos, SECTOR_SIZE);
          size -= SECTOR_SIZE;
          buffer += SECTOR_SIZE;
          reading_part += SECTOR_SIZE;
          open_files[fd].pos += SECTOR_SIZE;
          sector_read++;
          if (open_files[fd].pos < MAX_FILE_SIZE) { // check if file read reach to maximum file size
              if (Disk_Read(child_inode->data[sector_read], sectorBuffer) < 0) {
                return -1;
              }
              reading_pos = sectorBuffer;
          } else {
            return reading_part;
          }

      } else {
          dprintf("... Reading BYTE  %d\n", open_files[fd].pos);
          memcpy(buffer, reading_pos, (size_t) size);
          reading_part += size;
          open_files[fd].pos += size;
          return reading_part;
      }
  }
  return reading_part;
  //return -1;
}
```

File Write

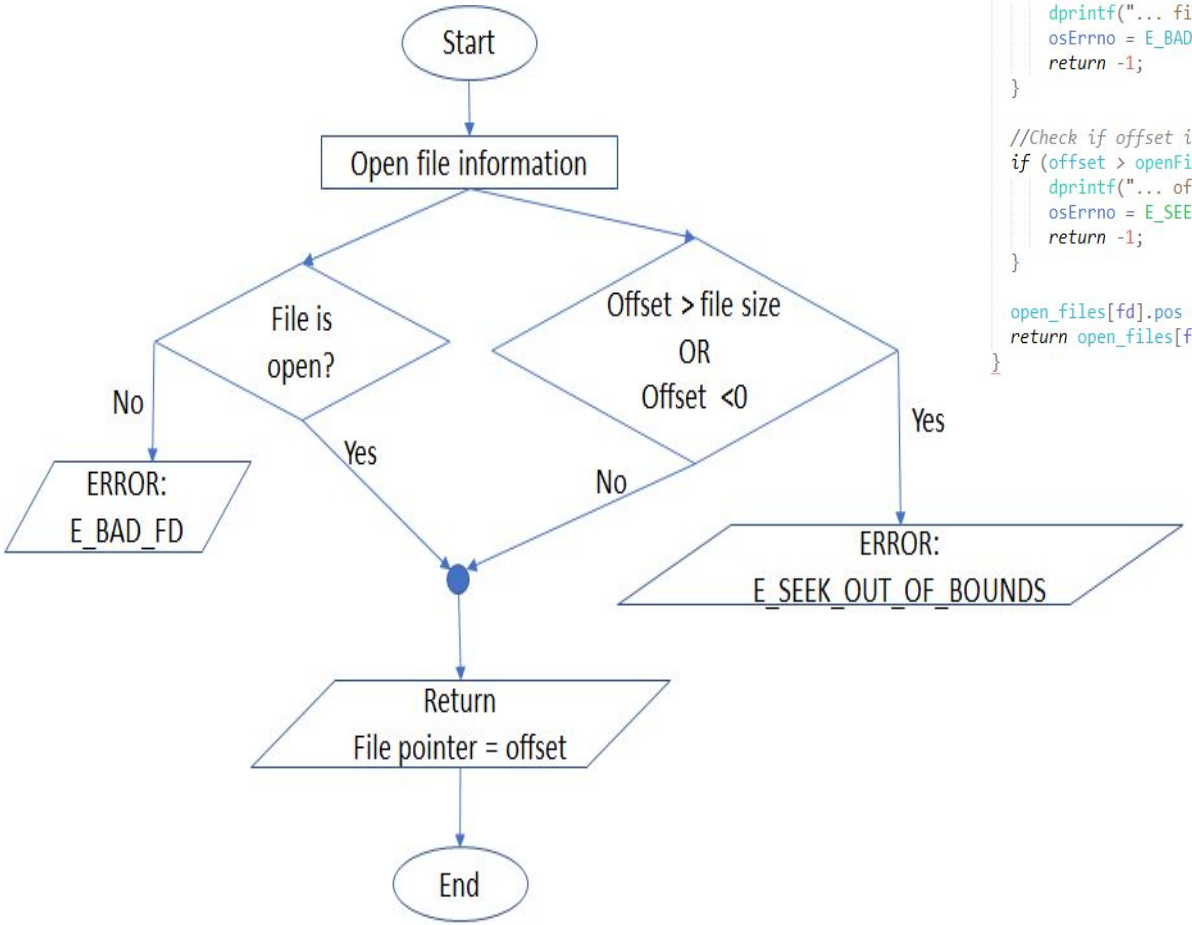
```
int File_Write(int fd, void* buffer, int size);
```




```c
/* YOUR CODE */
int File_Write(int fd, void* buffer, int size)
{
  int remaining_writing = size;
  // Have the information about the file (fd) where we have to write the buffer
  open_file_t openFileInfo = open_files[fd];
  // First check the file is open or not
  if (openFileInfo.inode == 0) {
      dprintf("... file not open");
      osErrno = E_BAD_FD;     // if not open set the osError to E_BAD_FD
      return -1;
  }

  // check if the file have enough space to write
  if ((openFileInfo.size + size) > MAX_FILE_SIZE) {
      dprintf("... file is too big.");
      osErrno = E_FILE_TOO_BIG;
      return -1;
  }

  // load the disk sector containing the inode
  int inode_sector = INODE_TABLE_START_SECTOR+ openFileInfo.inode/INODES_PER_SECTOR;
  char inode_buffer[SECTOR_SIZE];
  if(Disk_Read(inode_sector, inode_buffer) < 0) {
    osErrno = E_GENERAL;
    return -1;
  }
  dprintf("... load inode table for inode from disk sector %d\n", inode_sector);

  // get the inode
  int inode_start_entry = (inode_sector - INODE_TABLE_START_SECTOR) * INODES_PER_SECTOR;
  int offset = openFileInfo.inode - inode_start_entry;
  assert(0 <= offset && offset < INODES_PER_SECTOR);
  inode_t *fileInode = (inode_t *) (inode_buffer + offset * sizeof(inode_t));

  int sectors_need = (size / SECTOR_SIZE) + 1; // number of sectors needed to write the buffer

  // start the process of writting
  for (int i = 0; i < sectors_need; i++) {
      char sector[SECTOR_SIZE];
      int sector_ID = bitmap_first_unused(SECTOR_BITMAP_START_SECTOR, SECTOR_BITMAP_SECTORS, SECTOR_BITMAP_SIZE);

      // Check if there is any sector left for writing
      if (sector_ID < 0) {
          osErrno = E_NO_SPACE;
          dprintf("... no space left.");
          Disk_Write(INODE_TABLE_START_SECTOR + (openFileInfo.inode / INODES_PER_SECTOR), inode_buffer);
          return -1;
      }
      fileInode->data[i] = sector_ID;
      Disk_Read(sector_ID, sector);

      // write buffer to sector. Buffer might be bigger than sector
      // need to request more sectors
      if (remaining_writing > SECTOR_SIZE) {
          remaining_writing -= SECTOR_SIZE;
          memcpy(sector, buffer, SECTOR_SIZE);
          buffer += SECTOR_SIZE;
          fileInode->size += SECTOR_SIZE;
      } else {
          memcpy(sector, buffer, (size_t) remaining_writing);
          buffer += remaining_writing;
          fileInode->size += remaining_writing;
          remaining_writing = 0;
      }
      openFileInfo.size = fileInode->size;
      openFileInfo.pos = fileInode->size;
      Disk_Write(sector_ID, sector);
  }
  Disk_Write(INODE_TABLE_START_SECTOR + (openFileInfo.inode / INODES_PER_SECTOR), inode_buffer);
  return size - remaining_writing;
}
```

File Seek

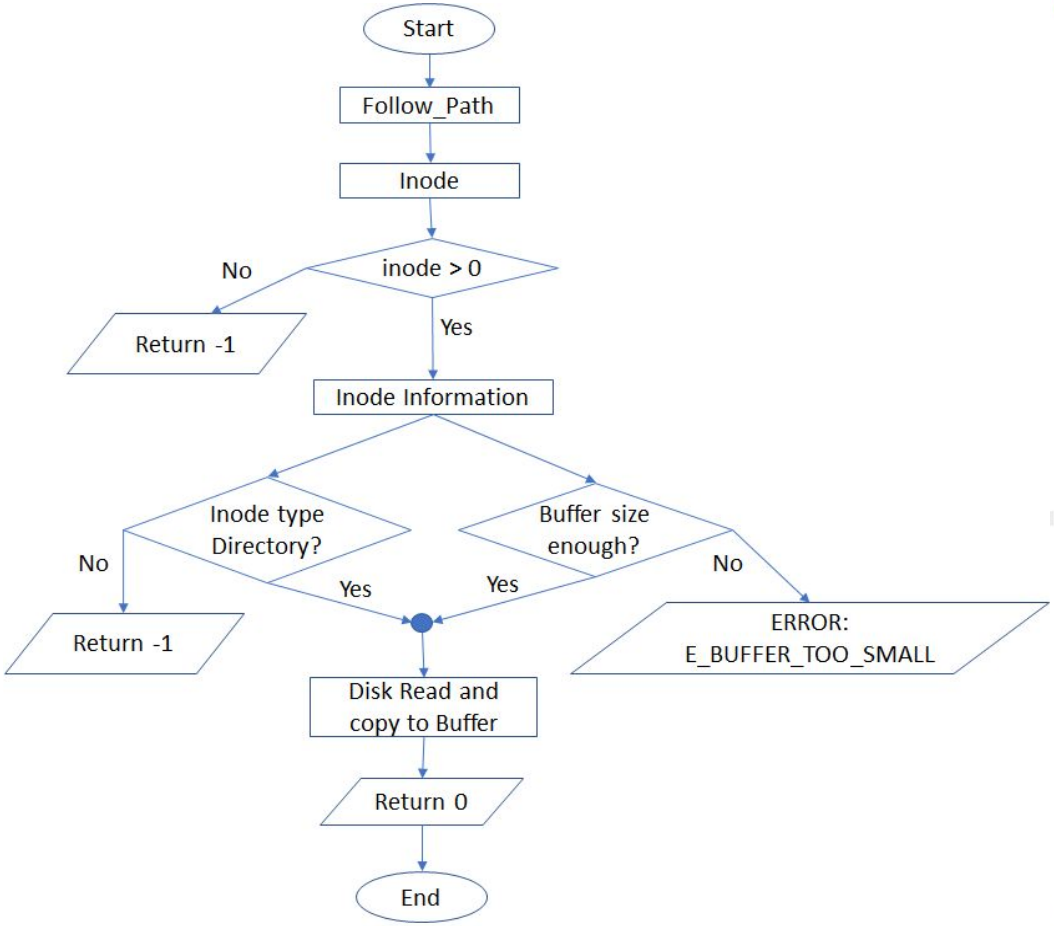
```
int File_Seek(int fd, int offset);
```




```
int File_Seek(int fd, int offset){
  open_file_t openFileInfo = open_files[fd];

  if (openFileInfo.inode == 0) { // check if the file is open or not
      dprintf("... file not open");
      osErrno = E_BAD_FD;
      return -1;
  }

  //Check if offset is larger than the size of the file or offset is negative
  if (offset > openFileInfo.size || offset < 0) {
      dprintf("... offset is out of bound");
      osErrno = E_SEEK_OUT_OF_BOUNDS;
      return -1;
  }

  open_files[fd].pos = offset;
  return open_files[fd].pos;
}
```

Directory Read

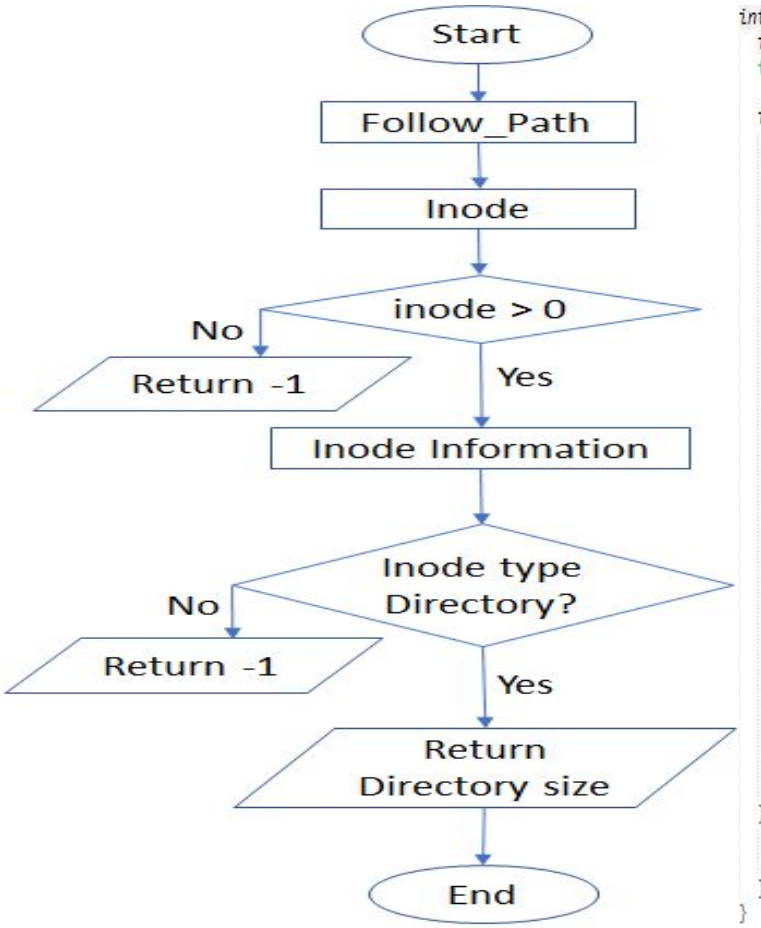
```
int Dir_Read(char* path, void* buffer, int size)
```




```
int Dir_Read(char* path, void* buffer, int size) {
  int parent_inode; //inode of the directory that need to read
  follow_path(path, &parent_inode, NULL);

  if (parent_inode >= 0) { //check the directory is found or not
    // Load the disk sector containing the inode
    int inode_sector = INODE_TABLE_START_SECTOR+parent_inode/INODES_PER_SECTOR;
    char inode_buffer[SECTOR_SIZE];
    if(Disk_Read(inode_sector, inode_buffer) < 0) {
      osErrno = E_GENERAL;
      return -1;
    }
    dprintf("... load inode table for inode from disk sector %d\n", inode_sector);


    // get the  inode
    int inode_start_entry = (inode_sector-INODE_TABLE_START_SECTOR)*INODES_PER_SECTOR;
    int offset = parent_inode-inode_start_entry;
    assert(0 <= offset && offset < INODES_PER_SECTOR);
    inode_t* directory_inode = (inode_t*)(inode_buffer+offset*sizeof(inode_t));
    dprintf("... inode %d (size=%d, type=%d)\n", parent_inode, directory_inode->size, directory_inode->type);


    // cheack weather the inode type is directory
    if(directory_inode->type != 1) {
      dprintf("... error: '%s' is not a Directory\n", path);
      osErrno = E_GENERAL;
      return -1;
    }

    //check if the read buffer is Larger enough to hold the details
    if(directory_inode->size*sizeof(dirent_t) > size){
      dprintf("ERROR: type-%d inode-%d size-%d givensize-%d\n", directory_inode->type, parent_inode, directory_inode->size, size);
      osErrno = E_BUFFER_TOO_SMALL;
      return -1;
    }

    int sector, i;
    int increase_size = 0; // increase the buffer size by this amount
    char dirent_buffer[SECTOR_SIZE];


    for(sector = 0; sector < MAX_SECTORS_PER_FILE; sector++){ //iterate over each sector
      if(directory_inode->data[sector]){
        Disk_Read(directory_inode->data[sector], dirent_buffer);
        for(i = 0; i < DIRENTS_PER_SECTOR; i++){ // for each directory read it and copy it to the buffer
          dirent_t* dirent = (dirent_t*)(dirent_buffer+i*sizeof(dirent_t));
          if(dirent->inode){
            memcpy(buffer+increase_size, (void*)dirent, sizeof(dirent_t));
            increase_size += sizeof(dirent_t);
          }
        }
      }
    }

    dprintf("%d\n", directory_inode->size);
    return directory_inode->size;
  } else {
    dprintf("... directory '%s' is not found\n", path);
    return 0;
  }

}
```

Directory Size

```
int Dir_Size(char* path)
```




```c
int Dir_Size(char* path){
  int parent_inode; //inode of the directory that need to read
  follow_path(path, &parent_inode, NULL);

  if (parent_inode >= 0) { //check the directory is found or not

    // Load the disk sector containing the inode
    int inode_sector = INODE_TABLE_START_SECTOR+parent_inode/INODES_PER_SECTOR;
    char inode_buffer[SECTOR_SIZE];
    if(Disk_Read(inode_sector, inode_buffer) < 0) {
      osErrno = E_GENERAL;
      return -1;
    }
    dprintf("... load inode table for inode from disk sector %d\n", inode_sector);


    // get the inode
    int inode_start_entry = (inode_sector - INODE_TABLE_START_SECTOR) * INODES_PER_SECTOR;
    int offset = parent_inode - inode_start_entry;
    assert(0 <= offset && offset < INODES_PER_SECTOR);
    inode_t *directory_inode = (inode_t *) (inode_buffer + offset * sizeof(inode_t));
    dprintf("... inode %d (size=%d, type=%d)\n", parent_inode, directory_inode->size, directory_inode->type);

    // cheack weather the inode type is directory
    if (directory_inode->type != 1) {
        dprintf("... error: '%s' is not a directory\n", path);
        osErrno = E_GENERAL;
        return -1;
    }
    dprintf("... RETURNING SIZE: '%d' \n", (int) (directory_inode->size * sizeof(dirent_t)));

    return (int) (directory_inode->size * sizeof(dirent_t));
  } else {
      dprintf("... directory '%s' is not found\n", path);
      return 0;
  }
}
```