
Project 3: Parsing

COP 4610 / CGS 5765
Principles of Operating Systems

Misc

- Email me your new team information
- Will **assign** teams if necessary after this Friday

Extract Values from BIOS Parameter Block (BPB)

- Bytes per sector
- Sector per cluster
- Number of FATs
- ...

Use Case

- **BPB_BytsPerSec**
 - ❑ Offset – 11
 - ❑ 2 bytes (little endian)
 - ❑ Count of bytes per sector. This value may take on only the following values: 512, 1024, 2048 or 4096.
- **Function call?**
 - ❑ `ParseInteger(offset, bytes, [valid values])`
 - Returns integer at offset
 - ❑ C++
 - `ParseInteger<uint16_t, 512, 1024, 2048, 4096>(base_addr+11);`

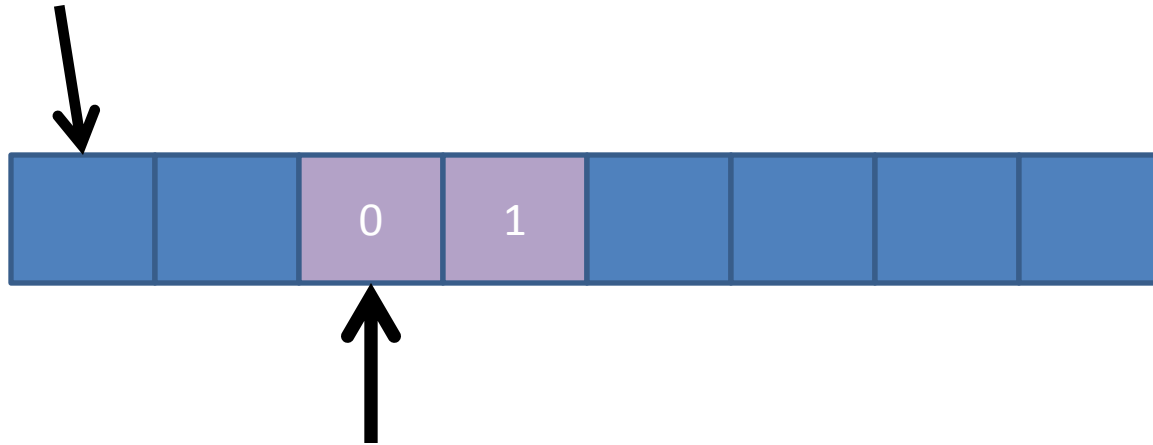
mmap()

```
auto fd = open("fat32.img", O_RDONLY);
if (fd < 0) {
    cerr << "error opening file" << endl;
    exit(EXIT_FAILURE);
}

int offset = 0;
unsigned len = 4096;
auto fdata = (uint8_t*)mmap
    (0, len, PROT_READ, MAP_PRIVATE, fd, offset);
```

Little Endian Source

`mmap(img)`



$$\text{uint16_t} = 0$$

$$\text{uint16_t} | = 1 \ll 8$$

C++ CODE

ParseInteger

```
unsigned int ParseInteger(const uint8_t* const ptr, int nr_bytes)
{
    int val = 0;
    for (size_t i=0; i<nr_bytes; ++i) {
        val |= static_cast<unsigned int>(ptr[i]) << (i*8);
    }
    return val;
}
```


ParseInteger

```
template<typename T>
T ParseInteger(const uint8_t* const ptr)
{
    T val = 0;

    for (size_t i=0; i<sizeof(T); ++i) {
        val |= static_cast<T>(static_cast<T>(ptr[i]) << (i*8));
    }

    return val;
}

ParseInteger<uint16_t>(fdata + 11);
```

ParseInteger

```
ParseInteger<uint16_t, 512, 1024, 2048, 4096>(fdata + 11);
```

```
template<typename T, const T...tArgs>
T parseInteger(const uint8_t* const ptr)
{
    T val = 0;
    for (size_t i=0; i<sizeof(T); ++i) {
        val |= static_cast<T>(static_cast<T>(ptr[i]) << (i*8));
    }

    constexpr auto valid_vals = initializer_list<T>({tArgs...});

    if (valid_vals.size() == 0) return val;

    for (auto &x: valid_vals)
        if (x == val) return val;

    throw exception();
}
```

ParseInteger

```
parseInteger<uint8_t>(fdata + 13, [] (uint8_t v) {return v!=0;})

template<typename T, typename...V>
T parseInteger(const uint8_t* const ptr, V&&...args)
{
    T val = parseInteger<T>(ptr);

    for (auto j: {args...})
        if (!j(val)) {
            stringstream ss;
            ss << "error parsing value" << val << endl;
            throw runtime_error(ss.str());
        }

    return val;
}
```

FAT32 DATA ARRANGEMENT

Root Directory

- Clusters
 - Numbered starting at 2
- First cluster contains the root directory
 - Locate using BPB values



Root Directory

`FirstDataSector`

`= BPB_ResvdSecCnt + (BPB_NumFATs * BPB_FATSz32);`

`RootClusterSector`

`= ((BPB_RootClus - 2) * BPB_SecPerClus) + FirstDataSector;`



usually 2

Records

- Describe directory contents (e.g., files/subdirs)
- 32-bytes
- 4 types
 - Unused
 - First byte is **0xE5**
 - End of directory
 - First byte is **0x00**
 - Short filename
 - Long filename text

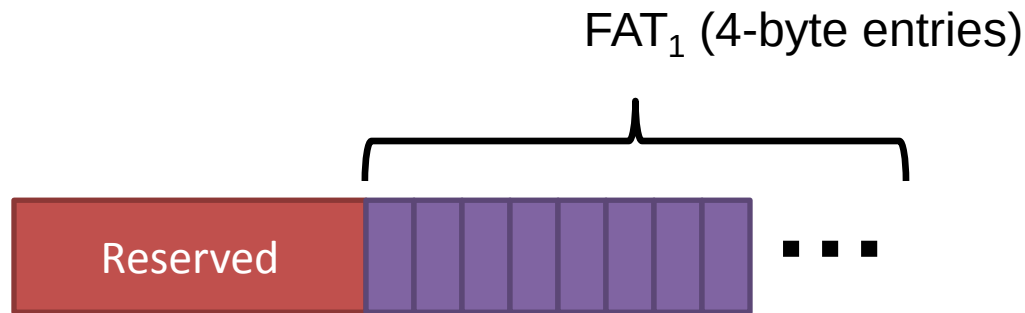
Records

- Short filename
 - **Name** (11-bytes)
 - **Attrib** (1-byte)
 - Is subdirectory, read-only, ...
 - **Cluster** (4-bytes)
 - Divided into high and low portions (2 bytes each)
 - **Size** (4-bytes)
- Long filename text
 - Discuss later...

FAT

- Needed if file/dir size is less than one cluster
- Stores location of the next cluster
- Array of 4-byte unsigned integers
 - Index into FAT with valid cluster number
 - Value at index is the next cluster number

Indexing into a FAT



`N = valid cluster number`

`FATOffset = N * 4;`

`ThisFATSecNum = BPB_ResvdSecCnt + (FATOffset / BPB_BytsPerSec);`

`ThisFATEntOffset = remainder(FATOffset / BPB_BytsPerSec);`

At This Point

- Read-only access
- You should be able to
 - Traverse directories and list all files
 - Output data in file
 - Use FAT for large directories/files
- Next step
 - Free space (records and clusters)
 - Find
 - Allocate

Getting Started

- Root directory
 - Find using values from BPB
 - List files/dirs in root directory
 - Parse records
- Design
 - BPB
 - Cluster data **uses (requires)** a BPB
 - Cluster **has** records

Next Steps

- Writing
 - Create files/dirs
 - Store data in file
- Allocation
 - Records
 - Clusters

Bonus

- Increase/decrease the number of FATs
- Defragment
- Visualize data layout
- fsck
 - Mark as bad
 - Copy data
- ...