COMP 3430

Operating Systems

July 8th, 2019

By the end of today's lecture, you should be able to:

- Describe the concept of an **address space** (chapter 13, in-class)
- Justify the layout of an address space (chapter 13, in-class)
- Justify who or what is responsible for managing segments (in-class)
- Describe the concept of paging. (chapter 18, in-class)

goal noun

\ ˈgōl , *chiefly Northern US especially in senses 3b and 2a also* ˈgül\

Definition of *goal*

1 : the end toward which effort is directed : AIM
// The *goal* is high-speed rail travel.

2 a : an area or object toward which players in various games attempt to advance a ball or puck and usually through or into which it must go to score points

b : the act or action of causing a ball or puck to go through or into such a goal

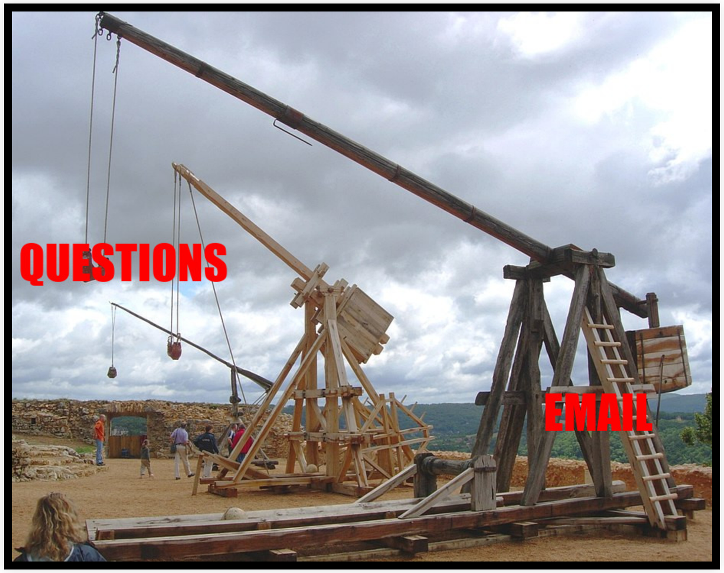
c : the score resulting from such an act

3 a : the terminal point of a race

b : an area to be reached safely in children's games

© Merriam-Webster

Questions by e-mail



Original © Luc Viatour (CC BY-SA 3.0)

> *My question is if there is an easy way to just fill the structs variables like bit by bit until it is full with the cluster (since they will be ordered correctly with pack), or do we have to actually manually assign each of the variables bytes as we read through the cluster? (read 16 bits assign struct.BPB_BytesPerSec = buf[16] … and then so forth for every single variable)*

Read the bytes in, then *cast* the bytes to the `struct` type.

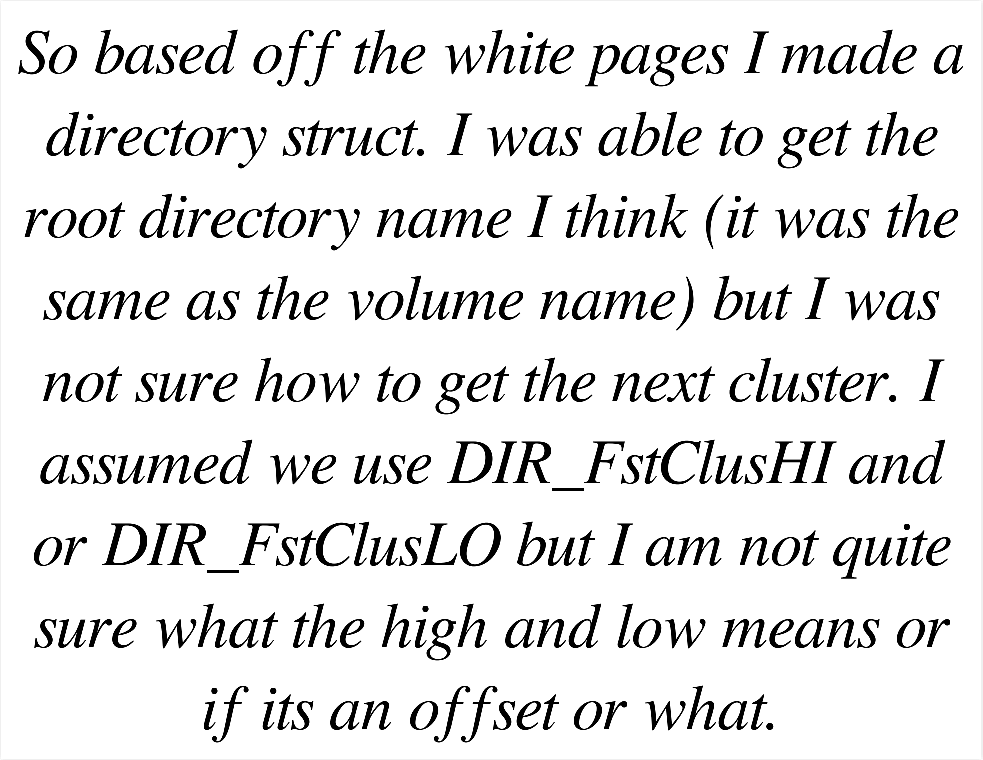
> *Now I have a couple of questions for the info part, I’ve been having trouble finding what specifically the drive name is? (or is this the name stored in BS_OEMName “mkfs.fat”) Is our free space supposed to be the exact same number as the bytes free value while using mdir? (mine is currently a little bit off)*

I use `BS_OEMName` from BPB. I don’t know how `mdir` is implemented, but it’s trivial to find out.

Is the drive name the same as the volume label? Should we be validating that things have the correct signatures before we do anything?

They *should* be, but I used the BPB for the drive name (see previous question).

Yes, validating the signatures is essential to confirming for yourself that you’re reading the right bytes.



So based off the white pages I made a directory struct. I was able to get the root directory name I think (it was the same as the volume name) but I was not sure how to get the next cluster. I assumed we use DIR_FstClusHI and or DIR_FstClusLO but I am not quite sure what the high and low means or if its an offset or what.

The root directory doesn't *have* a name (it *sort* of does, but it's just `\`).

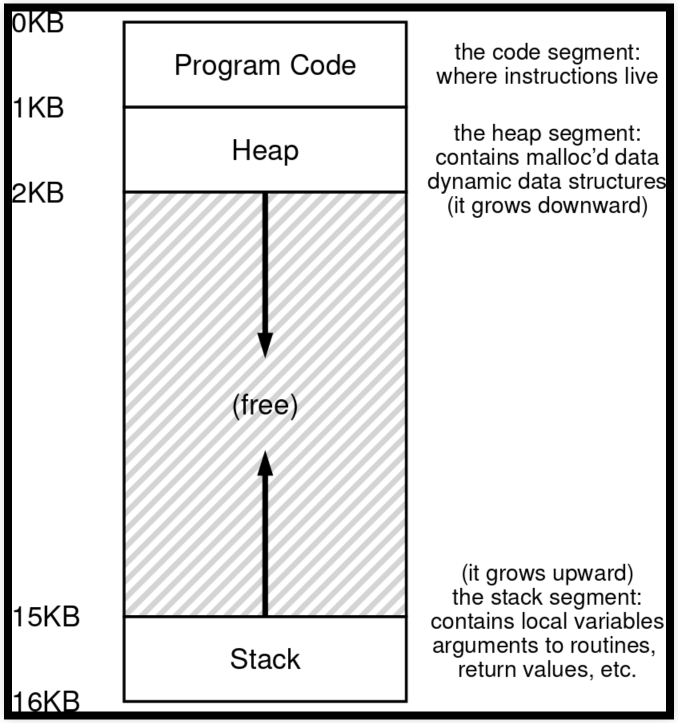
The `DIR_FstClusHI` and `DIR_FstClusLO` fields are two 2-byte values that should be stitched together to form a 4-byte (32 bit) number representing the first cluster number.

> *So is each cluster either a directory or file, so could you essentially just go from cluster to cluster and do the appropriate action?*

A cluster is neither a file or a directory, as far as we’re concerned a cluster is just data. We need the directory entries from a parent (which are coincidentally stored as data in clusters) to know how to interpret a chain of clusters (either as file data or as more directory entries).

Address spaces

- A running program *thinks* that its memory is laid out in a certain way.
  - Can we *observe* this?
  - Let’s take a look at `addresses.c`
  - Let’s look at `/proc/self/maps`

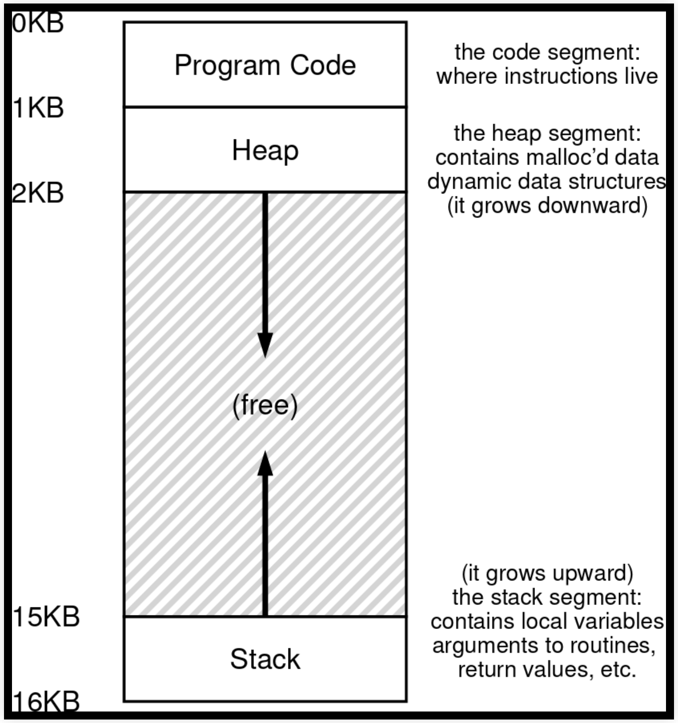


© RH & AC Arpacie-Dusseau

Think about it ⌛: Address spaces

Why might it be organized **this** way?

- What benefits are there for the OS?
- What benefits are there for the programmer?



© RH & AC Arpacie-Dusseau

Responsibilities

Think about it ⌛: Based on what you’ve read so far (and other courses like COMP 2280), who or what has the responsibility of managing the *contents* of:

1. The **code segment**?
2. The **stack**?
3. The **heap**?

Pixabay License

Managing the heap

Let’s try to *experimentally* determine if the OS manages the heap.

- Check out `allocate.c`
- Use `strace` to see what system calls are used.

Pixabay License

Heap allocators

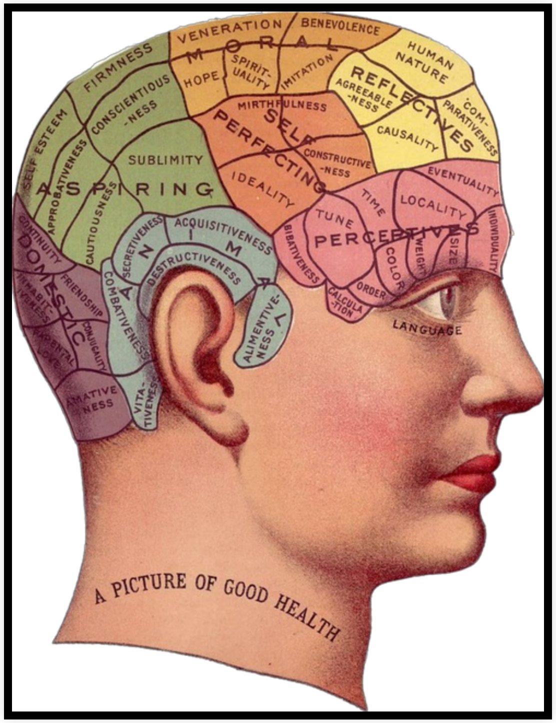
Here are *some* heap allocators:

1. `jemalloc`
2. `mimalloc`
3. ~~libc `malloc`~~

Investigate: Pick one allocator and find *a* system call that this allocator might use to ask the OS to work with the heap.

Managing memory

- The OS *does* have a responsibility to manage memory ✔
  - … but maybe not *quite* what we originally thought.



Pixabay License

Lower level management

Memory management by the OS is at a much lower level than “heaps” and “stacks”.

- **Main** responsibilities the OS has:
  1. Managing *physical* memory allocations
  2. Translating *virtual* addresses to physical addresses (with the hardware’s help)
- Hiding implementation details of the memory from a user process (abstraction!)

Paging

- … so it turns out that segmentation is *not* an ideal solution.
- **Paging** is a more granular generalization of segmentation
  - **Idea**: Treat memory like *many small segments*.
  - (also, it turns out that fine-grained segmentation wasn't such a bad idea after all…)

Pixabay License

You should be able to:

- Describe the concept of an **address space** (chapter 13, in-class)
- Justify the layout of an address space (chapter 13, in-class)
- Justify who or what is responsible for managing segments (in-class)
- Describe the concept of paging. (chapter 18, in-class)

goal noun

\ ˈgōl , *chiefly Northern US especially in senses 3b and 2a also* ˈgül\

Definition of *goal*

1 : the end toward which effort is directed : AIM
// The *goal* is high-speed rail travel.

2 a : an area or object toward which players in various games attempt to advance a ball or puck and usually through or into which it must go to score points

b : the act or action of causing a ball or puck to go through or into such a goal

c : the score resulting from such an act

3 a : the terminal point of a race

b : an area to be reached safely in children's games

© Merriam-Webster

Dumplings imply the
Existence of one large
Dumple