Week 7 - Friday

CS222

Last time

- What did we talk about last time?
- Allocating multi-dimensional arrays
- Random numbers

Questions?

Project 3

Quotes

In theory, theory and practice are the same. In practice, they're not.

Yoggi Berra

Rules for random numbers

- Include the following headers:
  - **`stdlib.h`**
  - **`time.h`**
- Use **`rand() % n`** to get values between **`0`** and **`n – 1`**
- Always call **`srand(time(NULL))`** **before** your first call to **`rand()`**
- Only call **`srand()`** **once** per program
  - Seeding multiple times makes no sense and usually makes your output much **less** random

Example

- Dynamically allocate an 8 x 8 array of `char` values
- Loop through each element in the array
  - With 1/8 probability, put a `'Q'` in the element, representing a queen
  - Otherwise, put a `' '` (space) in the element
- Print out the resulting chessboard
  - Use | and – to mark rows and columns
- Print out whether or not there are queens that can attack each other

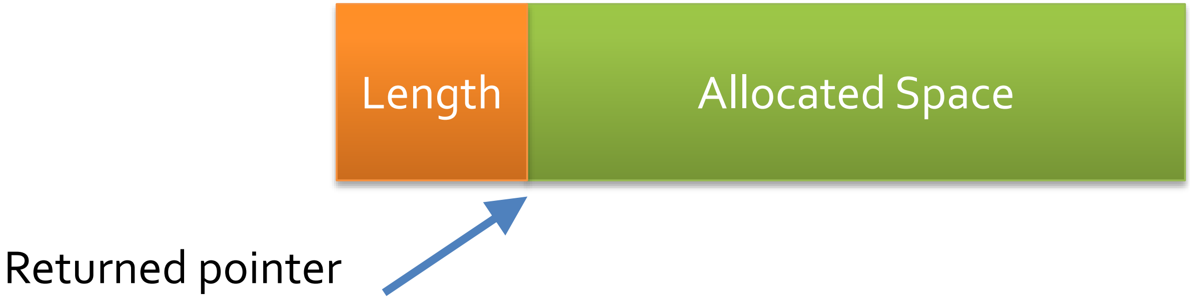
Memory Allocation (System Side)

Memory allocation as seen from the system

- There are really low level functions **`brk()`** and **`sbrk()`** which essentially increase the maximum size of the heap
- You can use any of that space as a memory playground
- **`malloc()`** gives finer grained control
  - But also has additional overhead

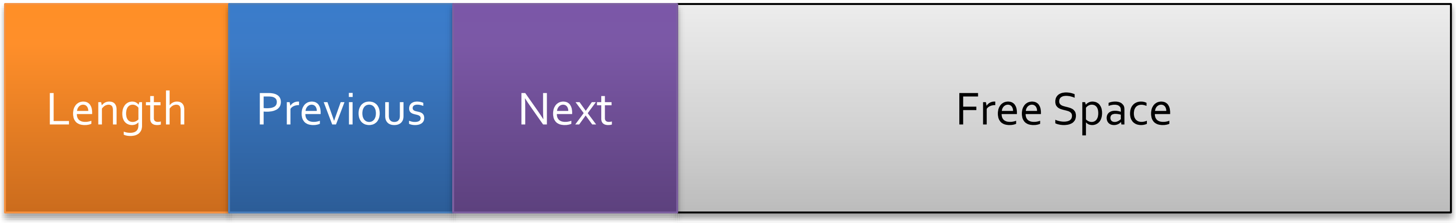
How does `malloc()` work?

- `malloc()` sees a huge range of free memory when the program starts
- It uses a doubly linked list to keep track of the blocks of free memory, which is perhaps one giant block to begin with
- As you allocate memory, a free block is often split up to make the block you need
- The returned block knows its length
  - The length is usually kept **before** the data that you use



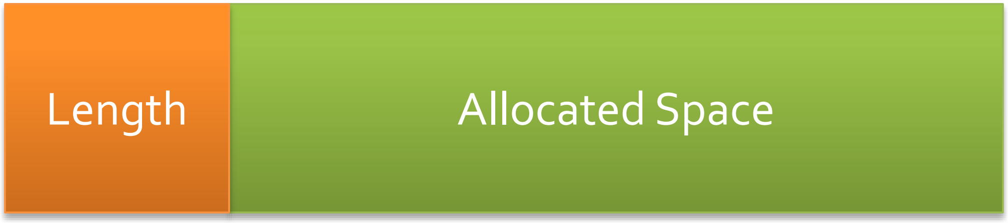
Free and allocated blocks

- The free list is a doubly linked list of available blocks of memory
- Each block knows its length, the next block in the list, and the previous block
- In a 32-bit architecture, the length, previous, and next data are all 4 bytes
    - Free block



Length | Previous | Next | Free Space

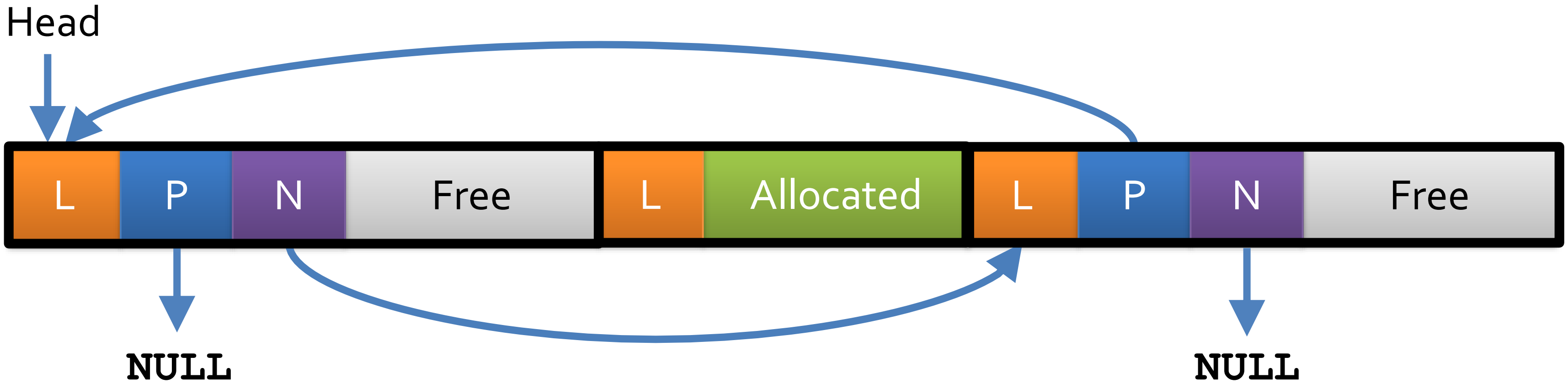
    - Allocated block



Length | Allocated Space

Free list

- Here's a visualization of the free list
- When an item is freed, most implementations will try to coalesce two neighboring free blocks to reduce fragmentation
  - Calling **`free()`** can be time consuming

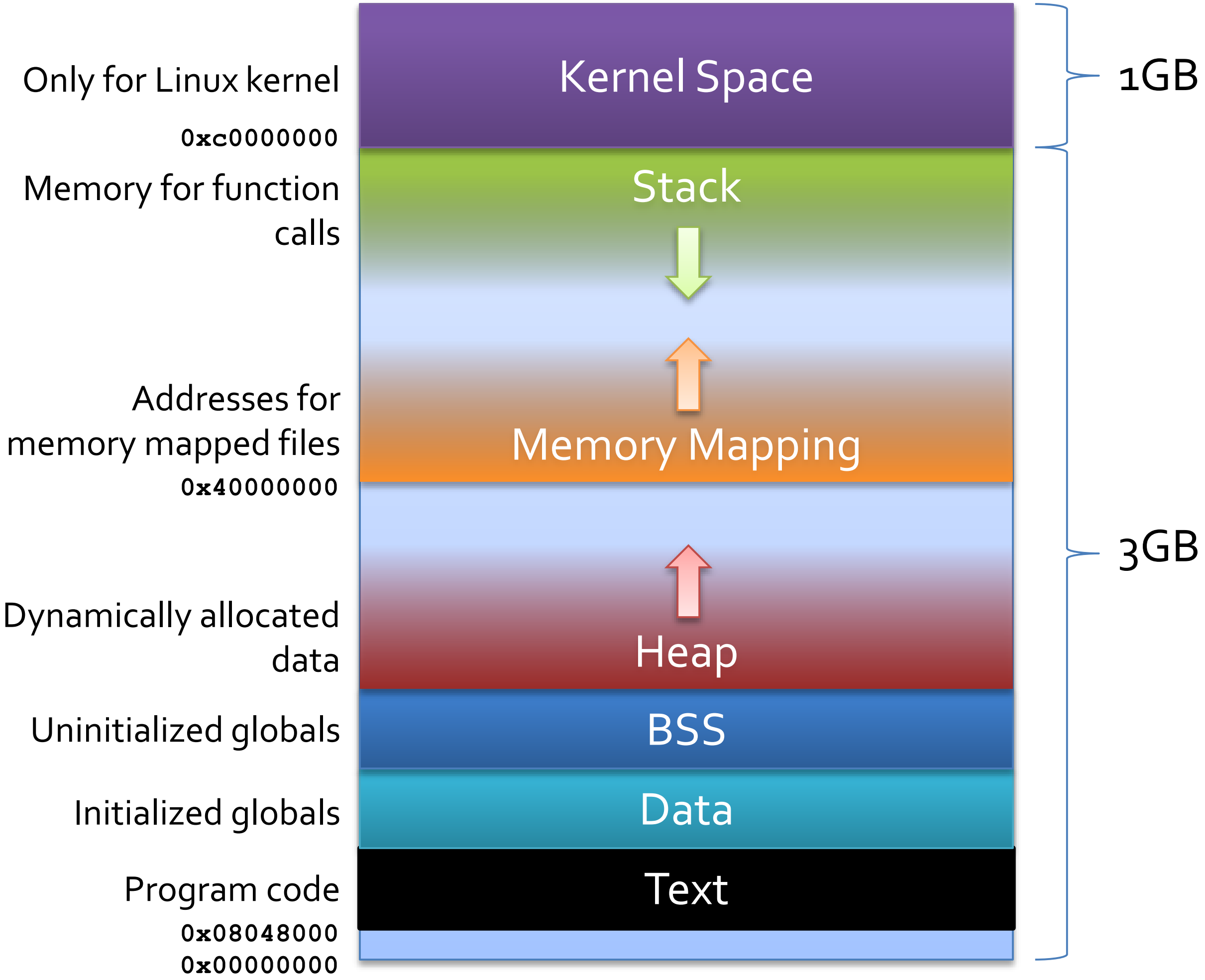


Other memory functions

- **`void* calloc(size_t items, size_t size);`**
  - Clear and allocate **`items`** items with size **`size`**
  - Memory is zeroed out
- **`void* realloc(void* pointer, size_t size);`**
  - Resize a block of memory pointed at by pointer, usually to be larger
  - If there is enough free space at the end, realloc() will tack that on
  - Otherwise, it allocates new memory and copies over the old
- **`void* alloca(size_t size);`**
  - Dynamically allocate memory on the stack (at the end of the current frame)
  - Automatically freed when the function returns
  - You need to **`#include <alloca.h>`**

Process memory segments

- Layout for 32-bit architecture
  - Could only address 4GB
- Modern layouts often have random offsets for stack, heap, and memory mapping for security reasons



Why aren't I showing the 64-bit version?

- The Linux machines in this lab use 64-bit processors with 64-bit versions of Ubuntu
- Our version of **gcc** supports 64-bit operations
  - Our pointers are actually 8 bytes in size
- But 64-bit stuff is confusing
  - They're still working out where the eventual standard will be
  - 64-bit addressing allows 16,777,216 terabytes of memory to be addressed (which is far beyond what anyone needs)
- Current implementations only use 48 bits
  - User space (text up through stack) gets low 128 terabytes
  - Kernel space gets the high 128 terabytes

Let's see those addresses

```
#include <stdio.h>
#include <stdlib.h>


int global = 10;

int main()
{
     int stack = 5;
     int* heap =
     (int*)malloc(sizeof(int)*100);
     printf("Stack:  %p\n", &stack);
     printf("Heap:   %p\n", heap);
     printf("Global: %p\n", &global);
     printf("Text:   %p\n", main);
     return 0;
}
```

Lab 7

Upcoming

Next time…

- Software engineering

Reminders

- Finish Project 3
  - Due by midnight tonight
- Have a good Spring Break!