Memory and C Programming

Robbert van Renesse
What is Memory?
What is Memory?

- I can’t recall...
I remember now:

• Memory is an array of bytes
• An index into this array is called an "address"
• A variable holding an address is called a “pointer”
Types of memory

• Code: machine instructions (read-only)
• Read-only data (string constants etc.)
• Global variables
• Heap: dynamically allocated memory
• Stack

You can store your data in global variables, on the heap, or on the stack
Logical view of process memory

- text: read-only text segment contains code and constants
- data: data segment contains global variables
- heap: heap used for memory allocation (malloc)
- stack: call stack

How many bits in an address for this CPU?
Why is address 0 not mapped?
Review: stack (aka call stack)

```c
int main(argc, argv){
    ...
    f(3.14)
    ...
}

int f(x){
    ...
    g();
    ...
}

int g(y){
    ...
    \[\text{PC/IP}\]
    ...
}
```

- **Arguments (3.14)**
- **Return address**
- **Saved FP (main)**
- **Local variables**
- **Saved registers**
- **Scratch space**
Review: heap

- **"break"**
- **"free list"**
- **pointer to next free chunk**
- **start of heap segment**
- **end of data segment**
- **in use**
- **free**

NULL
Three types of data memory

<table>
<thead>
<tr>
<th></th>
<th>Global</th>
<th>Heap</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocated</td>
<td>at start of process</td>
<td>using malloc()</td>
<td>at start of function call</td>
</tr>
<tr>
<td>initial state</td>
<td>as specified or 0 otherwise</td>
<td>junk</td>
<td>as specified or junk otherwise</td>
</tr>
<tr>
<td>released</td>
<td>at end of process</td>
<td>using free()</td>
<td>at end of function call</td>
</tr>
</tbody>
</table>
C Programming

• Like Java programming, but
  • no garbage collection
  • no type safety
  • no object-orientation, polymorphism, container types, ...

• Instead:
  • "structs" (instead of classes)
  • pointers (instead of references)
  • malloc/free (instead of new and garbage collection)
Hello World

#include <stdio.h>

int main()
{
    printf(“Hello World\n”);
    return 0;
}
struct square
{
    int width, height;
};

typedef struct square square_t;
Pointers

```c
void f()
{
    square_t sq1, sq2;    // on the stack!
square_t* ptr = &sq1;  // a pointer

    ptr->width = 300;
...
```
void f()
{
    square_t* ptr = malloc( sizeof(square_t) );

    ptr->width = 300;

    ...  

    free(ptr);

    ...
Project P0

• Implement a queue *and* a test program
• Has to be done by each student individually
  • by Wednesday February 8, so you have almost two weeks
  • but start today if only to find out how hard it is to get started on this!
    • you don’t want to find this out too late

• Tar file with instructions (README file) on CMS
On Testing

• Testing is at least as important as implementing the data structure itself
• Don’t be satisfied with a simple “it compiles and I can do a simple access”
  • think about corner case, and check those
    • dequeue from an empty queue
    • removing the first entry of a queue
    • removing the last entry of a queue
    • ...
Linux...

• All projects should be done in a Linux / x86 environment

• Learn to use
  • C compiler: cc (or gcc or clang)
  • Debugging tools: valgrind, gdb
  • Code project tools: make
  • Code repo: git
    • use private github repo on github.coecis.cornell.edu

• Running Linux
  • Can install Linux on your laptop
    • dual boot or virtual machine (Vmware, VirtualBox, Hyper/V, …)
  • Remote access
    • CSUGlab machines