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

- **Stack:** call stack
- **Heap:** heap used for memory allocation (malloc)
- **Data Segment:** data segment contains global variables
- **Text Segment:** read-only text segment contains code and constants

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

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

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

int g(int y) {
    ...  
    return address  
    ...  
}
```

- stack frame for `main()`
- stack frame for `f()`
- stack frame for `g()`

- 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
- NULL

- in use
- free
## Three types of data memory

<table>
<thead>
<tr>
<th></th>
<th>Global</th>
<th>Heap</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>allocated</strong></td>
<td>at start of process</td>
<td>using malloc()</td>
<td>at start of function call</td>
</tr>
<tr>
<td><strong>initial state</strong></td>
<td>as specified or 0 otherwise</td>
<td>junk</td>
<td>as specified or junk otherwise</td>
</tr>
<tr>
<td><strong>released</strong></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"
  • pointers
  • malloc/free
Hello World

```c
int main()
{
    printf("Hello World\n");
    return 0;
}
```
Structs

```c
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 next Friday 6pm, so you have one week
• Tar file with instructions (README file) on CMS