**CMPSCI 377 Operating Systems** **Fall 2005**

# Lecture 14: November 10

*Lecturer: Emery Berger* *Scribe: Eric Hodge, Eric Patrick*

Today:

- Memory Management

## 14.1 Memory Management

### 14.1.1 Introduction

- Not memory management in kernel, but memory management between app and OS - the "run time system."
- Java runs in virtual machine (in run time system.)
- C/C++ runs in libraries (libc.so, libc++.so)
- Explicit memory management (c / c++)
- Garbage collection (Java, Python, Perl)

### 14.1.2 Explicit Memory Management

- One of the oldest fields in computer science
- Must say explicitly what you want to do with memory (ask for it.)
- Malloc (size) - returns a pointer to space big enough for size bytes.
- Calloc (size, times) - multiplies size * times, also fills memory with zeros.
- Realloc (old obj, size) - reallocates old object to a chunk of memory of size.
- realloc (null, sz) = malloc(sz)
- Realloc ( p, 0) = free(p)
- Min size returned by malloc = 8 bytes, sizeof (double) ==8 bytes
- Free(ptr) - dispose of object.
- Takes object at ptr and gives it back to runtime system.
- If you don't free your objects, you get a memory leak.
- Things slow down due to paging.

realloc(NULL, size) == malloc(size)
realloc(p,0) == free(p)

### 14.1.3 Errors Involved in Memory Management

Dangling Pointer Error:
P = malloc()
x=p

...
...
Free(p)
Z = malloc()
z...
x...

z may have overwritten x
-you had a pointer to some space
-but now you've freed it
-and now it can be overwritten
-you can still try to reference it without no guarantees

Buffer overflow
- allocating too small a space and overwriting the end of memory block.
- Used by h4X0Rz.
- Professor Berger is l33t.

Some other errors...
- free objects that you didn't allocate
-free objects twice

### 14.1.4 Memory Allocation

What malloc() actually does:

- Process is instantiated.
- Loader (ld.so in linux) loads program to memory, and points program counter to right place and begins running.

Memory Structure:

- Stack grows down.
- Heap grows up.
- Code text segment beneath heap.
- In between stack and heap is a protected page to prevent collision between stack and pointer, is fixed.
- One way of managing heap size is to use a breakpoint (sbrk(int) to set pointer.)

mmap():
- mmap() often maps a file to memory.
- Most UNIXs have a file called /dev/zero.
    - Anonymous file.
- When calling mmap(), allocates memory in swap file for mmap() call.
- Munmap(ptr, sz) deallocates.

### 14.1.5 Issues in Memory Management

- Should not use an sbrk() and mmap() approach, only use mmap().
- Sbrk() only allows you to move breakpoint for heap. Mmap() allows you to remap all the heap to decrease heap size.

### 14.1.6 How memory manager actually manages memory

- Mmap() a big chunk of memory. Start and end pointers are at beginning.
- Call malloc(8), move end pointer to 8 bytes.
- Moving pointer is referred to as pointer bumping.

### 14.1.7 Freeing Objects

- Find a way to deallocate x in x,y,z.
- Cannot move objects around.
- Deallocate x, marked as being free.
- Header / Boundary Tag small amount of space at each section of memory to store object size and status (free or allocated.)
- First-Fit algorithm - On new malloc(), look for FIRST block that has not been allocated that object fits in.
    - Runs in worst case O(n).
    - Expected case O(n/2)
- Best fit algorithm - go through all of memory using linear search to find smallest chunk available with greater than required size.
    - Runs in O(n).
- Splitting - breaks free area of memory into smaller chunks to allow other smaller chunks of free memory to maximize utilization.
- Coalescing - joining adjacent chunks of memory together to fit larger objects.
- Linux allocator has a pointer to prevoius memory chunk and pointer to next.
    - Steals a bit for 0=free, 1=allocated.
- Since O(n) is bad, must manage memory differently.

### 14.1.8 Free Lists

- Organize array into sizes of chunks.
- Since there are many sizes, use size classes, which generalize some sizes into one array spot.
- When freeing an object, put object into free list under its size class and mark it as free.
- If requesting an object allocation, go to its size class and check for available chunk.
- If not available, take next largest (if available) OR advance to next object.
- Internal fragmentation - extra space allocated that will not be used because your object is smaller than allocated section.
- External fragmentation - space lost in between objects that is unusable as it is broken between objects.
- When calling malloc(), put lock around malloc()

- This prevents race condition from accessing same place in memory
- To avoid having threads slow down program, have multiple free lists.

### 14.1.9 Garbage Collection

- No such thing as free() method.
- Find all unused objects and deallocate them.
- Garbage collection tests for reachability.
- Roots  Globals, stack, and registers.
- Use roots to find pointers, then find more pointersetc.
- Build reachability tree. If there is no pointer to an object, it is unreachable and is garbage.
Use mark-sweep. Everything is initially garbage.
- For every object in tree, set mark bit to 1 when it is reachable.
- When done searching tree, sweep through heap, and deallocate all garbage.
- Garbage collector is called complete if it is guaranteed to reclaim all memory.
- Stop-the-world garbage collector stops program during garbage collection.

### 14.1.10 Semi-space collector

- Known as copying garbage collector.
- Divide heap in two.
- Once 1st heap is filled, run garbage collection.
- Look at roots and see what gets pointing to from roots.
- If it IS pointed to, copy to 2nd heap.
- Deallocate first heap.
- Then 2nd heap becomes from space.
- Generations - allocate to nursury.
- If object survives, copy out. If not, reset nursery.