Memory Management

CST 357/457 – Systems Programming
Michael Ruth, Ph.D.
Associate Professor
Computer Science & I.T.
mruth@roosevelt.edu

Objectives

- Discuss memory management concepts including overall process space usage
- Explain the process of allocating memory, freeing memory, and manipulating memory

Memory Management Concepts

- Memory is among the most basic and most essential resources of a process
- In general, the problems with memory management are about managing the resources we have not needing more
- Process Address Space:
  - UNIX (as with most OS) virtualize memory.
  - Processes do not directly address memory
  - The kernel provides each process with a unique virtual address space
    - Address space is linear (0 -> *) and flat (not divided)

Pages & Paging

- Memory is composed of bits, that form bytes, that form words, that form ***pages***
  - Machine architecture determines ***page size***
- A process cannot necessarily access all the pages on a system (they may not correspond to them)
  - Pages are either valid or invalid
    - Valid -> associated with an actual page of data in memory or **secondary storage**
      - Accessing an invalid page results in a segmentation violation
      - If it's in secondary storage, it results in a page fault
        - » The kernel intervenes and go gets it (paging in)
        - » If it swaps something out of MM (paging out)

Memory Regions

- The kernel arranges pages into blocks that share certain properties
  - Text segment
    - Contains program code, constants, string literals, and other read-only data
  - Stack
    - Processes execution stack which grows and shrinks, contains local variables, function return data, etc.
  - Data segment (or heap)
    - Contains processes' dynamic memory, malloc() and can grow and shrink as necessary
  - Bss segment
    - Contains uninitialized global variables

Allocating Dynamic Memory

- We did this already, but we'll go into detail… basically allocating memory
  - `#include <stdlib.h>`
  - `void * malloc(size_t size);`
- C' returns a **void pointer** by default, so we typically need to cast to the correct type
  - `name = (char *) malloc(512);`
- Malloc can/will return NULL, so it's always important to check and handle error conditions

Freeing Dynamic Memory

- Memory allocated with *lloc calls must be returned to the system when no longer in use via free()
  - `#include <stdlib.h>`
  - `void free(void * ptr)`

Manipulating Memory

- There are a number of functions provided for manipulating raw bytes of memory
  - Many of them are similar in operation to string functions

Setting Bytes

- `#include <string.h>`
- `void * memset(void *s, int c, size_t n);`
  - A call sets the n bytes starting at s to the byte c.
  - EX:
    - `memset(s,'\0',256)`

Comparing Bytes

- Compare two chunks of memory for equivalence:
  - `#include <string.h>`
  - `Int memcmp(const void *s1, const void *s2, size_t n)`
  - Compares the first n bytes of s1 and s2 and returns 0 if they're equal
    - Negative if s1 is less than s2
    - Positive if s1 is more than s2

Moving Bytes

- Copy the first n bytes from src to dest
  - `#include <string.h>`
  - `void * mmove(void *dst, const void *src, size_t n)`
  - Can handle overlapping memory segments
- If we do NOT want the src/dest to overlap (a safer/faster option):
  - `#include <string.h>`
  - `void * memcpy(void *dst, const void *src, size_t n)`
  - `void * memcpy(void *dst, const void *src, int c, size_t n)`
    - `Stops if it find a byte c`

Summary

- Discussed memory management concepts including overall process space usage
- Explained the process of allocating memory, freeing memory, and manipulating memory

Questions?