Basic memory management 1

CS503: Operating systems, Spring 2019

Pedro Fonseca
Department of Computer Science
Purdue University

Admin

- Grades on blackboard
- Lab 1 due next week
- Pay attention to Piazza for announcements, clarifications, etc.

Previous lecture(s)

- Process coordination:
  - Spin locks, semaphores, conditional variables
- Inter-process communication
  - Implemented by message passing
    - Can be synchronous or asynchronous
  - Synchronous interface is the simplest
  - Xinu uses synchronous reception and asynchronous transmission

Types of memory management

- Low-level memory management:
  - Manages memory within kernel address space
  - Allocates address space for processes
  - Treats memory as a single, exhaustible resource

Types of memory management

- High-level memory manager
  - Manages pages within address space
  - Divides memory into abstract resources

Conceptual uses of a low-level memory manager

- Allocation of “stack space” for a process:
  - Performed by the process manager
  - Primitives needed to allocate and free stack space
- Allocation of “heap storage”
  - Performed by device manager (buffers) and other system facility
  - Primitives needed to allocate and free heap space

Possible allocation strategies

- Allocation pool:
  - Large memory area reserved for allocation
- Free-list
  - A data structure connecting the free (available) space in a pool
- How to pick an allocation block from free-list
  - First-fit
  - Best-fit

Possible allocation strategies (cont.)

- Multiple layers within a pool:
  - Per-size
  - Per-process free-list
  - Per-thread free-list
  - Cache layer
- Allocator examples in practice:
  - Slab allocator (FreeBSD/Linux kernel)
  - Doug Lea's Malloc (User space)
  - TCMalloc (User space)

Practical considerations

- Sharing
  - Stack should not be shared b/w processes
  - Multiple processes may share heap
- Persistance
  - Stack object is associated with one process
  - Heap object may persist longer than the process that created it
- Avoiding unused memory:
  - Minimize internal/external fragmentation
- Efficient metadata manipulation/access
  - Fast identification of an available free block

Low-level memory manager in Xinu

- Stack allocation/free in Xinu

```
addr = getstk(numbytes);
freestk(addr, numbytes);
```

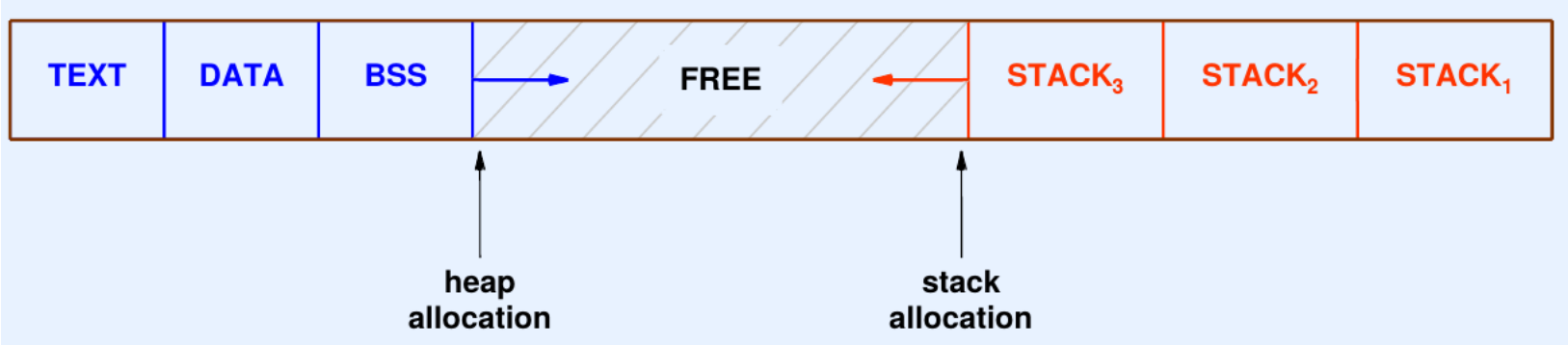
- Heap allocation/free in Xinu

```
addr = getmem(numbytes);
freemem(addr, numbytes);
```

Xinu low-level allocation

- One free area
- Single free list used for both heap and stack allocation:
  - Ordered by increasing address
  - Singly-linked
  - Initialized at system startup to contain all free memory
- Allocation policy
  - Heap: first-fit
  - Stack: last-fit
  - Results in two conceptual memory pools

Xinu approach



- First-fit allocates heap from lowest free address
- Last-fit allocates stack from highest free memory
- Uniform stack object size means higher probability of reuse

Memory protection and stack overflow

- If memory management hardware supports protection
  - Wrap the stack with non-accessible pages
  - Enforce hardware-based access control to non-accessible pages
- Q: What if no hardware protection available

Memory allocation granularity

- Memory is byte addressable
- Some hardware/instructions require alignment
- Cannot allocate/free individual bytes
- Difficult to migrate allocated objects at runtime
- Solution: choose minimum granularity and round all requests

Illustration of Xinu free list

- Free memory blocks used to store list pointers
- Items on list ordered by increasing address
- All allocations rounded to size of *struck memblk*
- Length is *memlist* counts total free memory bytes

Allocation technique in Xinu

- **getmem():**
  - Round up the requested allocation
  - Walk free memory list
  - **First-fit**: choose first free block that is large enough

Deallocation technique in Xinu (v2)

- **freemem():**
  - Round size to a multiple of metadata blocks
  - Walk the free list, using *next* to point to a block on the free list, and *prev* to point to the previous block (or *memlist*)
  - Stop when the address of the block being freed lies between *prev* and *next*
  - Either:
    - Coalesce with the previous block if new block is contiguous
    - Add the new block to the free list
  - Coalesce with the next block, if the result of the above is adjacent with the next block

Summary

- To preserve OS multi-level hierarchy, divide memory manager into two pieces:
  - Low-level used in kernel to allocate address spaces
  - High-level used to handle abstraction of virtual memory and paging within an address space
- Two conceptual memory types in low-level pieces
- Two may conceptual types of memory:
  - Heap
  - Stack