PURDUE UNIVERSITY®

CS 25200: Systems Programming

Lecture 7: malloc() and Memory Allocation Errors

Prof. Turkstra

Feasting with faculty

- Tuesdays at 12:00pm, Earhart Private Dining Room A
- Always welcome – no invitation needed

- Will still send out weekly batches of invitations
  - You don’t have to wait to get one!

Some notes

- Start projects early!
- Please read the posts on Piazza before asking a question
  - Many duplicate questions so far
- Reminder about code on Piazza
- Lab attendance
  - First hour required, attendance still taken
  - Second hour optional
  - Can attend second hour of any lab section
    - Priority to those actually enrolled

Lecture 07

- malloc() implementation notes
- Explict vs. implicit allocation
- Memory leaks
- Premature frees
- Wild frees
- Memory smashing
- Debugging

Project 2 Grading

- 100 points for project
  - 10 points checkpoint 1 – next Monday
  - 80 points provided test cases
  - 10 points “robustness” test cases
- 10 points for code standard
  - 1 point deducted for each subsection violated
  - Remember the linter *

  * Passing the linter does not mean you have fully adhered to the standard

Free list

- We will use a head pointer to keep track of the list
  - NULL when list is empty
- The list used in this lab will be doubly linked
  - …and NULL terminated (head's prev and tail's next)
- Be careful about how you manipulate the list – follow the specification
  - Insert at the head
  - When coalescing, reuse existing pointers as described

malloc()

- Round up requested sizes to the next MIN_ALLOCATION byte boundary
  - If size is 0, return NULL
- Add the size of the allocated header

```
real_size = roundup(requested_size, MIN_ALLOCATION);
```

- Ensure there is enough space for next/prev pointers when free()'d

```
real_size = real_size + ALLOC_HEADER_SIZE < sizeof(header) ?
            sizeof(header) - ALLOC_HEADER_SIZE: real_size;
```

- Use the specified algorithm to identify an appropriate block
- Split only if the remainder is large enough to accommodate an >= MIN_ALLOCATION byte request

- If split, remember to add the remainder to the free list
  - Insert at the head
- If a large enough block does not exist, request a new ARENA_SIZE chunk from the OS
  - Or a larger multiple
- Finally, return a pointer to the memory region immediately following the header
  - Hint: what is the address of the data field?

free()

- Obtain the left header, if free, coalesce
- Obtain the right header, if free, coalesce
- Don’t forget to update size in header and left_size in neighboring block
- When coalescing, the block should remain at the same position in the free list
  - Otherwise insert at the head

Fence posts

- Must avoid attempts to coalesce with memory below or above the heap
- Other libraries may call sbrk() as well, creating holes in the heap
- To prevent coalescing in these cases, a “dummy header” or fence post should be added to the beginning and end of a chunk
  - Single header (ALLOC_HEADER_SIZE) with state set to FENCEPOST

DL Malloc

- The standard memory allocator used by (g)libc is Doug Lea's malloc
- You can see it here:
  - http://g.oswego.edu/dl/html/malloc.html

Purdue trivia

- "Harry Creighton Peffer, first head of the School of Chemical Engineering, ordered a camera through normal university channels. Purchasing Agent H. C. Mahin wrote to ask Peffer what he intended to do with it. The implications of Mahin's query so angered Peffer that he fired back one of the best letters he ever regretted. It was, Peffer wrote to Mahin in high dudgeon, 'none of your goddam business' what he did with the camera, but since Mahin wanted to know, he intended to take pictures with it." The letter eventually made its way to President Elliott's desk. "Peffer was called to the president's office where Elliott told him that writing such a letter to an administrative officer was the same thing as writing it to him. Peffer was unimpressed. He pointed out to Elliott that he too was an administrative officer and that using the same logic, Elliott could conclude that he had written the letter to himself."
  'Peffer, what am I going to do with you?' the exasperated president asked.
  'I haven't the slightest idea,' Peffer retorted.
  'Get out of here and get back to your office,' Elliott said with finality."

  - A Century and Beyond, by Robert W. Topping

Memory allocation

- Explicit memory allocation is generally faster than implicit memory allocation
  - Explicit: malloc(), free()
  - Implicit: garbage collection

Explicit memory allocation

- Can be error prone…
  - Memory leaks
  - Premature frees
  - Double free
  - Wild frees
  - Memory corruption (smashing)

Memory leaks

- Allocated and unused objects are not freed when they should be
- Process' memory usage continually increases
  - Eventually spills into swap
  - May cause the system to hang or halt
  - Or, OOM Killer

- Memory leaks are often more pronounced when you have a long-running process
- Some (poorly written) servers need to be “rebounced” (restarted)
- Short lived programs may get away with memory leaks
  - Memory is always freed when processes terminate
- “Slow but persistent disease”

Example

```
int *myint = malloc(sizeof(int) * 2);
...use myint[] ...
// old myint is leaked
myint = malloc(sizeof(int) * 4);
...use myint[] ...
```

- Or…

```
while (1) {
  void *ptr = malloc(100);
}
```

Premature frees

- Object continues to be used after free() is called
- Freed object has been added to the free list
  - Next/previous pointers are set, corrupting the first 8-16 bytes of the object
- Modifying the beginning of the object will corrupt the next/previous pointers
- Difficult to debug – problems may arise far away from the original error

Example

```
int *p = (int *) malloc(sizeof(int));
*p = 8;
free(p);
...
*p = 9;
...
int *q = (int *) malloc(sizeof(int));
```

Always nullify

```
int *p = (int *) malloc(sizeof(int));
*p = 8;
free(p);
p = NULL;
...
*p = 9;
```

Double free

- Multiple calls to free() for the same object or address
- Object may end up added to the free list multiple times, corrupting it
- Future calls to malloc()/free() may crash

Example

```
int *p = (int *) malloc(sizeof(int));
...
free(p);
...
free(p);
```

Wild frees

- Calling free() on something that was not originally malloc()'d
- No header
- May crash on free()
- ...or future calls to malloc()/free()
- ...or somewhere else in the program

The obvious

- Memory allocated with malloc() should only be deallocated with free()
- Leave the rest alone

- Wild frees is also referred to as “freeing non-heap objects”

Example

```
int q;
int *p = &q;
...
free(p);
```

```
char *p = (char *) malloc(1000);
p = p + 10;

free(p);
```

Memory smashing

- Program writes to memory outside of the intended data structure or allocated range
- Corrupts the following object(s)
  - In the heap, may overwrite boundary tags or even fencepost(s)
- Sometimes nothing happens
  - Why?

Example

```
char *s = malloc(8);
strcpy(s, “Hello, World!”);
```

Debugging memory allocation errors

- Can be very difficult
  - Corruption of malloc's internals may not become apparent until much later
  - Memory leaks may not be immediately obvious
- Trying to find premature/double/wild frees?
  - Can try commenting out all free() calls
  - If the problem goes away, uncomment them one-by-one

Debugging

- ...or use a tool
- Free
  - Valgrind
  - Memcheck
- Not free
  - IBM Rational Purify
  - Bounds Checker
  - Insure++

Tradeoffs

- Explicit memory allocation is...
  - Fast
  - Efficient with regard to memory usage
- It also...
  - Can be error prone
  - Requires more expertise
  - May take longer to develop and debug
  - Can be insecure if not done right

- Some prefer to use languages like Java or C#
- C/C++ is still widely used
  - Especially for lower-level, performance critical applications

Debugging allocators

- Can always augment your allocator to make it more robust
- Check if a block was returned by malloc() using a magic number in the header
  - Prevents wild free()s
- Use different magic numbers to indicate free/allocated
  - Instead of a single bit
  - E.g. free: 0xF7EEF7EE; allocated: 0xA10CA7ED
  - Always check the magic number

- Create an integrity checking function
  - Iterate over all free and allocated blocks
  - Check for two consecutive free blocks
  - Ensure there are no unexpected gaps
- Store size of gaps in the fence posts
- Verify sizes and flags in header match
- Print free and allocated blocks

Questions?