Debugging

15-213: Introduction to Computer Systems
Recitation 12: Monday, Nov. 9th, 2013

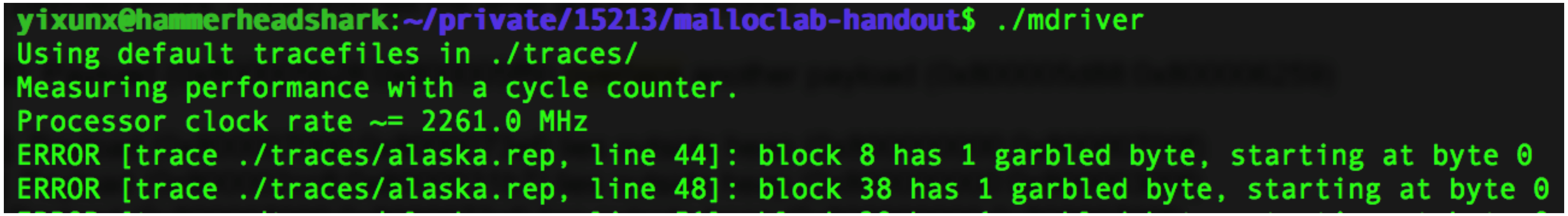
Yixun Xu
Section K

News

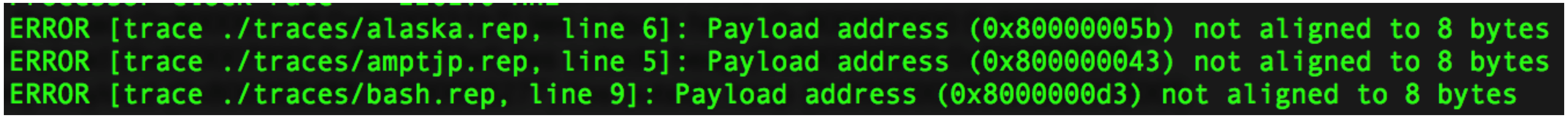
- Malloc Lab due Thursday Nov 14th

Errors

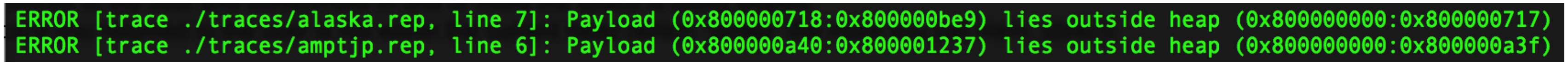
- Some errors are identified by the driver



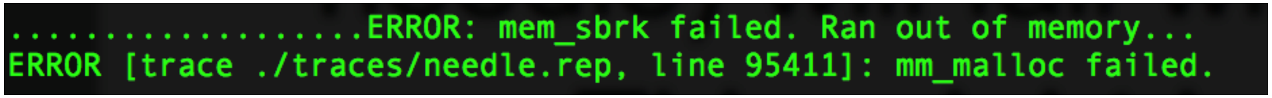
```
yixunx@hammerheadshark:~/private/15213/malloclab-handout$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2261.0 MHz
ERROR [trace ./traces/alaska.rep, line 44]: block 8 has 1 garbled byte, starting at byte 0
ERROR [trace ./traces/alaska.rep, line 48]: block 38 has 1 garbled byte, starting at byte 0
```



```
ERROR [trace ./traces/alaska.rep, line 6]: Payload address (0x80000005b) not aligned to 8 bytes
ERROR [trace ./traces/amptjp.rep, line 5]: Payload address (0x800000043) not aligned to 8 bytes
ERROR [trace ./traces/bash.rep, line 9]: Payload address (0x8000000d3) not aligned to 8 bytes
```



```
ERROR [trace ./traces/alaska.rep, line 7]: Payload (0x800000718:0x800000be9) lies outside heap (0x800000000:0x800000717)
ERROR [trace ./traces/amptjp.rep, line 6]: Payload (0x800000a40:0x800001237) lies outside heap (0x800000000:0x800000a3f)
```

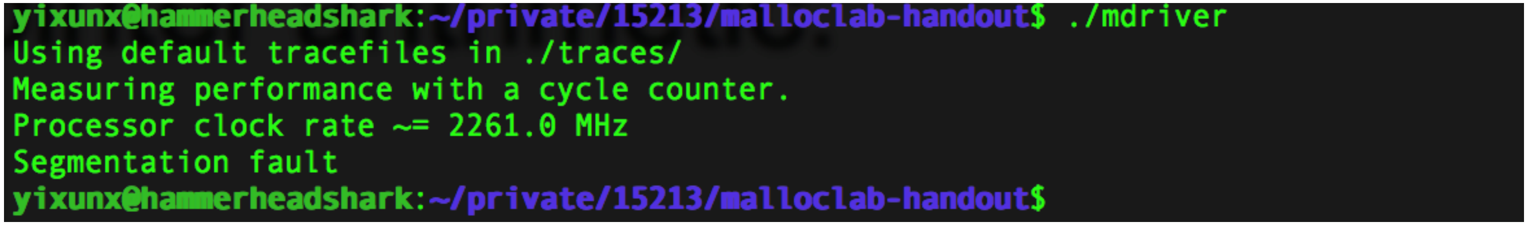


```
....................ERROR: mem_sbrk failed. Ran out of memory...
ERROR [trace ./traces/needle.rep, line 95411]: mm_malloc failed.
```

- The error message is straightforward in most cases
  - “garbled byte” means part of the payload returned to the user has been overwritten by your allocator
  - “out of memory” occurs when the memory is used very inefficiently, or there are lost blocks

Errors

- But most of the times…



```
yixunx@hammerheadshark:~/private/15213/malloclab-handout$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2261.0 MHz
Segmentation fault
yixunx@hammerheadshark:~/private/15213/malloclab-handout$
```

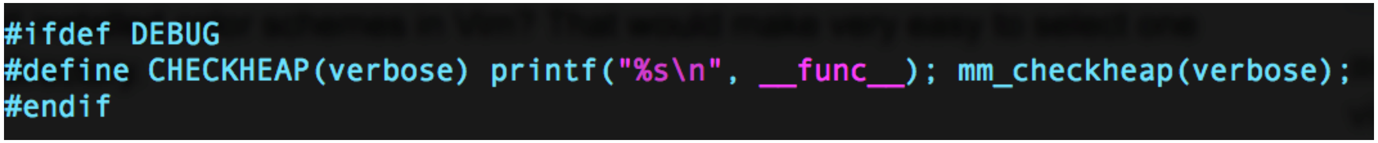
- Do “gdb mdriver” and “run” to find out which line segfaults
  - Note that a segfault occurring at line 200 could actually be caused by a bug on line 70

Segfault

- To resolve a segfault, it is necessary to find the earliest time things went wrong.
- One way to do this is to print the whole heap before/after relevant functions
  - Scroll up from the point of segfault and find the earliest operation that makes the heap look wrong
  - Sometimes this gives too much information, not all of which are useful
- The heap checker can make this easier
  - Checks violation of invariants (corruption of the heap)

Heap Checker

- Once you’ve settled on a design, write the heap checker that checks all the invariants of the particular design
- The checking should be detailed enough that the heap check passes if and only if the heap is truly well-formed
- Call the heap checker before/after the major operations whenever the heap should be well-formed
- Define macros to enable/disable it conveniently
  - e.g.



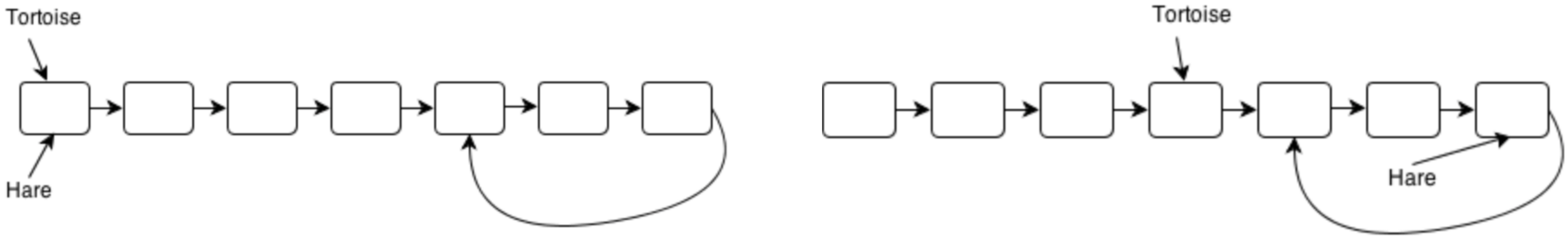
```
#ifdef DEBUG
#define CHECKHEAP(verbose) printf("%s\n", __func__); mm_checkheap(verbose);
#endif
```

Invariants (non-exhaustive)

- Block level:
  - Header and footer match
  - Payload area is aligned
- List level:
  - Next/prev pointers in consecutive free blocks are consistent
  - Free list contains no allocated blocks
  - All free blocks are in the free list
  - No contiguous free blocks in memory (unless you defer coalescing)
  - No cycles in the list (unless you use circular lists)
  - Segregated list contains only blocks that belong to the size class
- Heap level:
  - Prologue/Epilogue blocks are at specific locations (e.g. heap boundaries) and have special size/alloc fields
  - All blocks stay in between the heap boundaries
- And your own invariants (e.g. address order)

Hare and Tortoise Algorithm

- Detects cycles in linked lists
- Set two pointers “hare” and “tortoise” to the beginning of the list
- During each iteration, move the hare pointer forward two nodes and move the tortoise forward one node. If they are pointing to the same node after this, the list has a cycle.
- If the tortoise reaches the end of the list, there are no cycles.



Other things to watch for

- Uninitialized pointers and/or memory
- Make sure mm_init() initializes everything
  - It is called by the driver between each iteration of every trace
  - If something is overlooked, you might be able to pass every single trace file, but the complete driver test will fail

Useful Tools

- Valgrind
  - Illegal accesses, uninitialized values…
- GDB
  - watch, rwatch, awatch

Asking for help

- It can be hard for the TAs to debug your allocator, because this is a more open-ended lab
- Before asking for help, ask yourself some questions:
  - What part of which trace file triggers the error?
  - Around the point of the error, what sequence of events do you expect?
  - What part of the sequence already happened?
- If you can’t answer, it’s a good idea to gather more information…
  - How can you measure which step worked OK?
  - printf, breakpoints, watchpoints…

Asking for help

- Bring to us a detailed story, not just a “plot summary”
  - “Allocations of size blah corrupt my heap after coalescing the previous block at this line number...” is detailed
  - “It segfaults” is not
- Most importantly: don’t hesitate to come to office hours if you really need help

Beyond Debugging: Error prevention

- It is hard to write code that are completely correct the first time, but certain practices can make your code less error-prone
- Plan what each function does before writing code
  - Draw pictures when linked list is involved
  - Consider edge cases when the block is at start/end of list
- Write pseudocode first
- Document your code as you write it

Beyond Debugging: Version control

- “I had 60 util points just 5 minutes ago!”
- Save the allocator after each major progress
- Most basic: copy files around using the cp command
- Alternatively: keep different versions in separate c files, and use “ln –s mm-version-x.c mm.c” to start using a particular version
- Or use git/svn/cvs…
  - Make sure your repository is private if you use remote repos

Optimization

- To achieve better performance, sometimes you would want to tweak certain parameters.
  - Number of size classes, the separation of size classes, the amount by which the heap is extended (CHUNKSIZE)…
- It is better to write modular and encapsulated code so that changing the parameters only requires changing a few lines of code
  - Use macros wisely

Optimization

- When you hit a bottleneck, find which part is limiting your performance
- A profiler is good for this kind of job
- To use gprof:
  - Change the Makefile to add “-pg” to the compilation flag
  - Run the driver. This will generate a file called gmon.out
  - Run “gprof ./mdriver” to see the result
  - Don’t forget to change the Makefile back

Final Words

- Start now, if not already
- Come to office hours early
- Write the heap checker well
- Be prepared to start over several times
- Before handing in, check:
  - Does the header comment contain a detailed description of your approach?
  - Is the indentation correct? Any line over 80 chars? (go to autolab to verify these)

Questions?

- Good luck!