CacheLab Recitation Notes (TA Guide)

Timings:

Section	Timing
Intro/Logistics (Slides 1-7)	3 min
Cache Lab Overview/Hints	5 min
Cache Concepts	5 min
Activity 1: Traces	15 min
C Review	5 min (fast)
Activity 2: Getopt	Remaining time

Learning Objectives:

- Review cache concepts (locality, cache misses, organization, read & writes)
- Introduce students to the trace file format and have them practice writing a short trace
- Review basic c concept and point students to the bootcamp if they need more help
- Gain an understanding of man pages and get opt and valgrind

Important Notes

Cache concepts

- Ask if the students have any specific questions about caches. If they do, feel free to take some time to go over those concepts
- Go through the example with a “real” address and see how it breaks down

Trace writing activity:

- Objective for student
    - Understand the format of trace files
    - Observe how the trace files create certain behavior in a cache
- Show the trace file, empty cache, and memory. Have students draw on these things out on a separate sheet of paper and fill in if each line in the trace is a hit or a miss and how the cache changes (5 min)
- Go over the slides together
    - Make sure to ask the questions to check student understanding

C-Review

- Ask if the students have anything specific they want to go through. If they do, feel free to prioritize that over the prepared material. It would be a good idea to read through C-Bootcamp (happened on Sunday) in case they have specific questions from the bootcamp.
- The categories focused during this recitation are pointers, arrays, and structs / unions.

Valgrind Demo:

- Objective for student
    - Familiarize with basic tags to run valgrind
    - Observe subtle mistakes in the program that valgrind can expose
- Main idea:
    - The program is a basic input-output program that is meant to take a string of size-4
    - The input string is compared to “0123”, and based on comparison result prints out response
- Mistakes
    - First error in this program is that scanf adds an additional null-terminating character to the end of the string, so the allocated size should be 4+1
    - Second error is that the allocated space is not freed before returning
- Link: https://www.cs.cmu.edu/~213/activities/valgrind_demo.tar
- Instructions for running:
    - First, run it without valgrind by
        - gcc -std=c99 valgrind_demo.c -lm
        - ./a.out
    - Then run it with valgrind and explain the error message
        - valgrind --leak-check=full ./a.out

Getopt Example

- Pull up man pages on projector
- Instead of doing answer, you actually mention which parts of the man page you read, so students get a demo in reading man pages
- Explain how man pages work- useful source of info for description/arguments/return values/error codes etc. for functions + useful for future classes and industry
- Explain what getopt is + highlights (go over args etc.)
- Allow them to work on the getopt example activity for a couple minutes on their own
- Getopt man page highlights (in appendix):

- `int getopt(int argc, char * const argv[], const char *optstring);`
  - `int argc` → argument count passed to `main()`
    - Note: includes executable, so `./a.out 1 2` has `argc=3`
  - `char * const argv` is argument string array passed to `main`
  - `const char *optstring` → string with command line arguments
    - Characters followed by colon require arguments
      - Find argument text in `char *optarg`
    - `getopt` can't find argument or finds illegal argument sets `optarg` to "?"
    - Example: "`abc:d:`"
      - `a` and `b` are boolean arguments (not followed by text)
      - `c` and `d` are followed by text (found in `char *optarg`)
- Returns: `getopt` returns -1 when done parsing

Blocking activity (10/11):

- This is designed to help students gain a better intuition about blocking and why/how we do it
- Start with going through the regular method of performing matrix multiplication
  - DRAW OUT the cache and what's in it
  - Label hits and misses
  - Calculate overall hit/miss rate
- Then go through how multiplication is done with blocks
  - Again, draw out the cache and how what's in it changes
  - Label hits and misses
  - Calculate overall hit/miss rate
- Use comparison to show the advantage of blocking!
- Non blocking example: a miss = 50%, b miss = 100%
- Blocking example: a miss = 25%, b miss = 25%

Cache questions (10/11)

- Please make sure to go over these questions ahead of time so you understand the answer and can provide explanations!
- Give students time to talk with their neighbors before asking for an answer
- Try to encourage all students to think and participate rather than just a few more vocal students
- Consider drawing/writing out how you can figure out the answer
- What type of locality?
    - C) Both A & B
        - Array indices are being access sequentially in time (temporal) and by index (spatial)
    - C) Both A & B
        - Array indices are being access sequentially in time (temporal) and by index (spatial)
- Calculating Cache Parameters
    - D) 4
        - b = 4 gives that block size = 2^4 = 16
        - Since the size of an int is 4 bytes, the block can store 4 ints
- Direct-Mapped Cache Example
    - B) t=27, s=2, b=3
        - We have that there are 4 sets, which gives s=2, and blocks of size 8, which gives b=3
        - m = t+s+b ⇒ 32 = 2+3+t ⇒ t=27
- Which set is it?
    - D) 3
        - Same cache parameters as previous problem (given on bottom of slide)
        - 0xFA1C = 0b1111101000011100
        - s = 0b11 = 3
- Cache Block Range
    - D) 0xFA18 – 0xFA1F
        - Range of values with same tag and set index, but last 3 bits (block offset) can vary
- Cache Misses
    - C) 64
        - 16 ints, each with size 4 bytes
    - D) 50%
        - Each block stores 16 ints (64 bytes) and we access every 8
        - The first access is a miss and the second is a hit
    - D) 50%

Appendix

- Mention that there are appendix slides for this recitation which explain some questions in more detail as well as give more questions
- Also sides about style and using git and using fscanf