Scribe Notes (October 8th)
Ling-Ya Monica Chao

- Strided access (associative caches)
  - What is strided access?
    - Reading by jumps rather than sequentially
    - Still predictable, still easier for cache compared to totally random access
    - Stride-K access pattern: reads/writes ever k^{th} byte (stride-1 = sequential)
  - Examples
    - Multi-dimensional array
    - Strided access of a physical dictionary
    - An array of structs

      ```
      struct employee {
              char* name;
              ......
              double salary;
      }
      ```

      ---> strided access when calculate the average wage of employees
  - Why is running "l09/r13-stridebyte" faster than "l09/r15-stdiostridebyte"?
    - Ans: for every character that is required, stdio copies 4096 bytes into the buffer.
    - But not that much slower, why? Because it's not that much more expensive to copy 4096 bytes than 1 byte.
  - How can we improve the code for strided access?
    - Having many slots in the cache
    - An alternative (l09/r17-multistdiostridebyte): basically implementing an associative cache. Open multiple files at the same time, each having a cache, but in charge of different sections of the file ---> don't need to throw away caches too early.
      - Associative: multiple slots
      - Fully associative: multiple slots, each address can be freely placed in all the slots
      - Direct map: each address can only be placed in one specific slot
  - Another special case of access: l09/r19-stdioendsbyte
    - Read a chunk of memory in the front then jump to the end to read another chunk
    - The kernel always prepare to do sequential reading, but the application can give advice to the kernel for "explicit prefetching"
      - Prefetching: load into buffer before request
      - Explicit prefetching: continue to next section
- Explicit prefetching
  - Definition: "application" requests to do prefetching = give the kernel more notice.
  - Can be safely ignored, and wouldn't affect the progress of the program.
  - The ideal case is if we can notify the kernel even before reading the first chunk: use "posix_fadvise".
- Coherency

- Stdio cache: not coherent, even within the same file (reason: if two caches are not coherent with the kernel cache, there is no reason for them to be coherent of each other).
- Buffer cache: coherent

---------- BREAK----------

- Processor cache

Disk --- (OS) ---> Primary Memory --- (processor) ---> Level 3 cache --- (processor) ---> Level 2 cache --- (processor) ---> Level 1 cache --- (processor) ---> Register

  - Block sizes
    - Block size for buffer cache (in primary memory relative to disk): 4096 bytes
    - Block size for other caches (relative to primary memory): 64 bytes
    - Data size in the registers: 4 bytes
    - What determines the cache size? Ans: bigger because you can store more data (pattern predictable, fetching is slow, so should do more batching), but also need to consider that for unpredictable patterns, bigger caches increases the fetching time.
  - Let's look at l11/servicetranslate-01
    - Purpose of program: search for corresponding port given input
    - Why is it slow? Try "strace", and we can see that, for every search attempt, the entire database is read/loaded again, which takes a lot of time.
    - Let's try to rewrite this software --- by using cache
      - Definition of cache:

```
typedef struct scache_slot {
   struct servent entry;
} scache_slot;
```

      - Definition of servent:

```
struct servant {
        char* s_name;
        char* s_proto;
        int s_port;
}
```

    - Attempt 1: l11/servicetranslate-03

```
struct servent* my_getservbyname(const char* name, const char* proto) {
  if (!scache)
    populate_services_cache();
  for (size_t i = 0; i < scache_size; ++i)
    if (strcmp(name, scache[i].entry.s_name) == 0
      && strcmp(proto, scache[i].entry.s_proto) == 0)
      return &scache[i].entry;
  return NULL;
}
```

        - Bad locality: the cache stores the pointers to the respective servents, but it's still random access when trying to obtain the information in them
- Attempt 2: l11/servicetranslate-01
    - Create a global for "tcp", so that all the pointers to the information "tcp" are pointed to the same piece of memory
    - Still slow? Swap the conditional, so that "slow circuit" will allow program to skip the second condition in an "and" statement when the first condition is false.