CS 354 - Machine Organization & Programming
Tuesday, October 15, 2019

Project p3 (6%): DUE at 10 pm on Monday, October 28th

Homework hw3 (1.5%): DUE at 10 pm on Friday, October 18th

Last Time

Placement Policies
Free Block - Too Large/Too Small
Coalescing Free Blocks
Footers

Today

Footers (from last time)
Explicit Free List (from last time)
Explicit Free List Improvements
Heap Caveats
Memory Hierarchy
Locality

Next Time

Designing Caches
Read: B&O 6.4 intro - 6.4.2

Explicit Free List Improvements

Free List Ordering

address order: order of blocks in mem from low → high address.

malloc with FF

+ Better mem util. than last-in order.

free

– slow O(N) must search free list for addr orders insert location.

last-in order: last (most recent) freed block is linked to front of free list.

malloc with FF

– Looks most recently freed block first

free

+ fast, constant time O(1), just link at free list end.

Free List Segregation

has multiple free lists – use an array of them
malloc chooses appropriate free list given the requested block size.

simple segregation: has one free list for each block size.

structure simple, no headers, blocks only need successors PTRs (pointers).

malloc chooses block at front of appropriate free list.

if list is empty: gets more heap from OS., divide into blocks, add to free list.

free Fast O(1), Link to appropriate free list.

problem fragmentation.

fitted segregation: has one free list for each size range.

+ mem util: as good as best fit.

+ Thruput: since search only part of heap (blocks of addr size).

fitting search of appropriate free list either first fit or best fit.

splitting puts new free block in appropriate free list.

coalescing puts new larger free block in appropriate free list of larger size range

Heap Caveats

Don't assume consecutive heap allocations result in contiguous payloads!

→ Why? ① payloads are interspersed with heap struct. and possible padding
② placement policies & heap struct can scatter allocations throughout the heap.

Don't assume heap memory is initialized to 0!

OS initially clear heap page to ∅ for security
but your recycle heap mem will have old data and structure info.

Do free all heap memory that your program allocates!

→ Why are memory leaks bad?
They slowly kill your program's performance by cluttering heap with garbage allocations and bad leaks could ultimately consume your heap

→ Do memory leaks persist when a program ends?
No.

Don't free heap memory more than once!

→ What is the best way to avoid this mistake? NULL freed pointer vars.

Don't read/write data in freed heap blocks!

→ What kind of error will result? intermittent error.

Don't change heap memory outside of your payload!

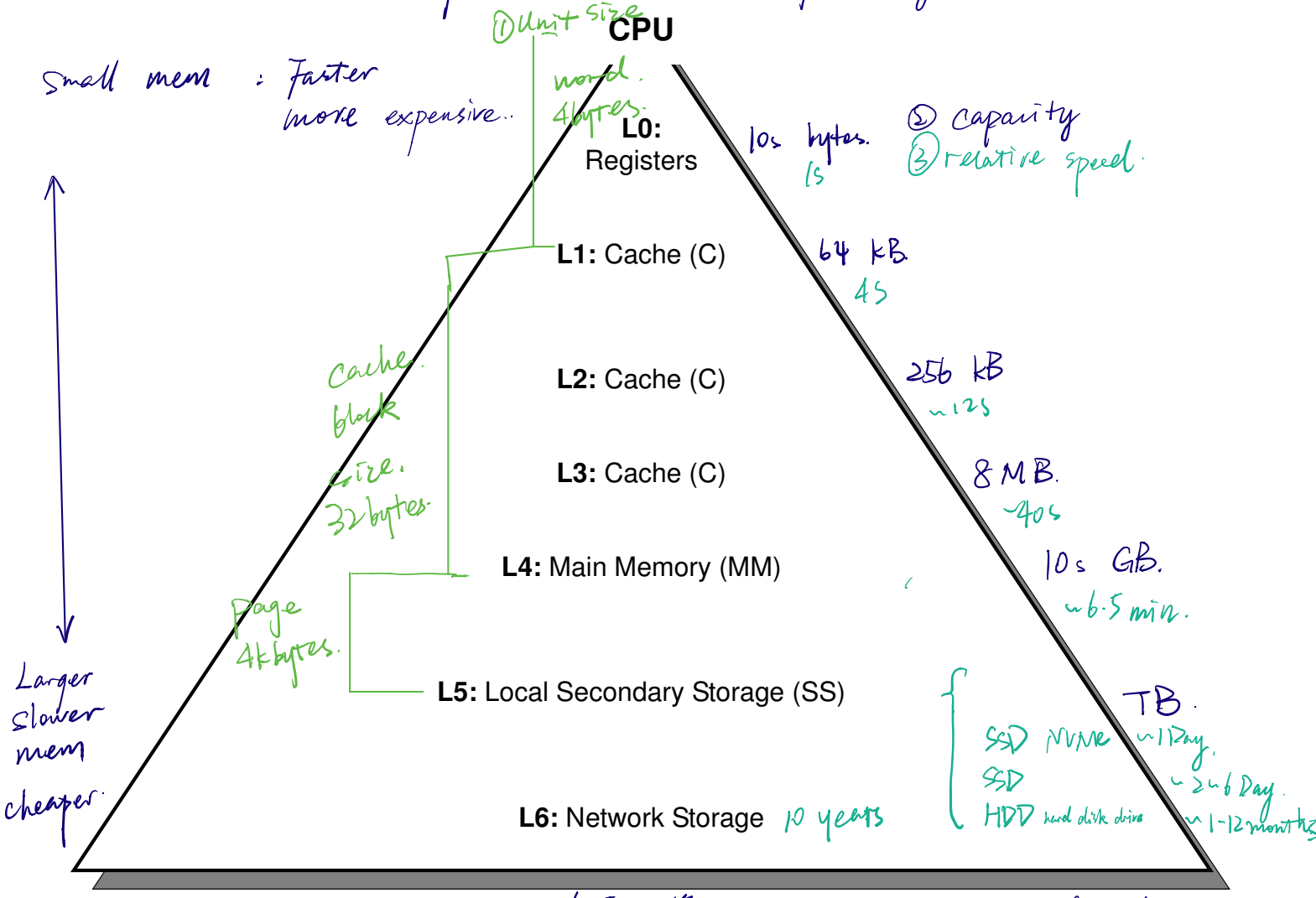
→ Why? Could trash the heap's internal structure and/or another block's payload.

Do check if your memory intensive program has run out of heap memory!

→ How? Check allocator's return value to ensure it is not NULL.

Memory Hierarchy

- *The memory hierarchy* gives the illusion to CPU of having lots of fast mem.



Cache is a smaller faster mem that acts like a staging area for data in a larger slower mem.

Memory Units

word: size used by CPU transfer between L1 & CPU

block: size used by C transfer between C Level & MM

page: size used by MM transfer between MM & SS

Memory Transfer Time

cpu cycles: used to measure time

latency: memory access time,

Locality

❋ *Programs with good locality* run faster since they work better with the mem hierarchy.

Why? Programs with good locality keep data that is likely to be accessed by the CPU at the top of the hierarchy mem is faster.

Locality designed to hardware – caching
OS – mem paging
APPs

What?

temporal locality: when a recently accessed memory location is repeatedly accessed in the near future.

spatial locality: when a recently accessed memory location is followed by near by memory locations being accessed in the near future.

Example

```
int sumArray(int a[], int size, int step) {
   int sum = 0;
   for (int i = 0; i < size; i += step)
      sum += a[i];
   return sum;
}
```

→ List the variables that demonstrate temporal locality.

i, size, sum, step

→ List the variables that demonstrate spatial locality.

a[i] if step is small enough.

stride: is step size measured in words returned sequential accesses

spatial locality = 1/stride.