CS 61 Lecture 16

Thursday, Oct 24

New Unit: Storage

memory, disk

SSD, hard drive

iCloud

cache

moving from

very simple model of memory

address ↔ read/write

↓

memory system

hierarchy of storage devices w/ different capacities, costs, and times to access

Analogy: Books

instead of data

desk	3 books	≡ registers	~16
shelves	60 books	≡ caches	
Widener	3.5M books	≡ main memory	32GB
Depository	10M books	≡ disk drive	256 GB
Borrow Direct	90M books	≡ network / cloud storage	

↓ hierarchy

- Why not connect the processor directly to main memory?

add A, B not possible, why? problems w/ costs and time to access

appx. .com				
DDR4 DRAM	16 GB	$400	$25/GB	12.5-15 ns
SSD/flash	256 GB	$200	$0.80/GB	0.1 ms
Hard Drive	2TB	$100	$0.50/GB	12 ms

volatile, erases when power off

non-volatile, still present when power off

KEY OBSERVATION:

CAN'T BUILD STORAGE THAT IS BIG, FAST, AND CHEAP ⇒ create a system w/ appearance of big and fast

HOW? books

① most likely to read

② things I generate (write)

data

Principle: locality of reference

- good (fast on modern machines) take advantage of locality of reference).

1. Temporal locality
   if referenced, likely to reference again soon
2. Spatial locality
   if referenced, those around it likely referenced

```
long f(const int* v, int n) {
    long sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += v[i];
    }
}
```

temporal locality

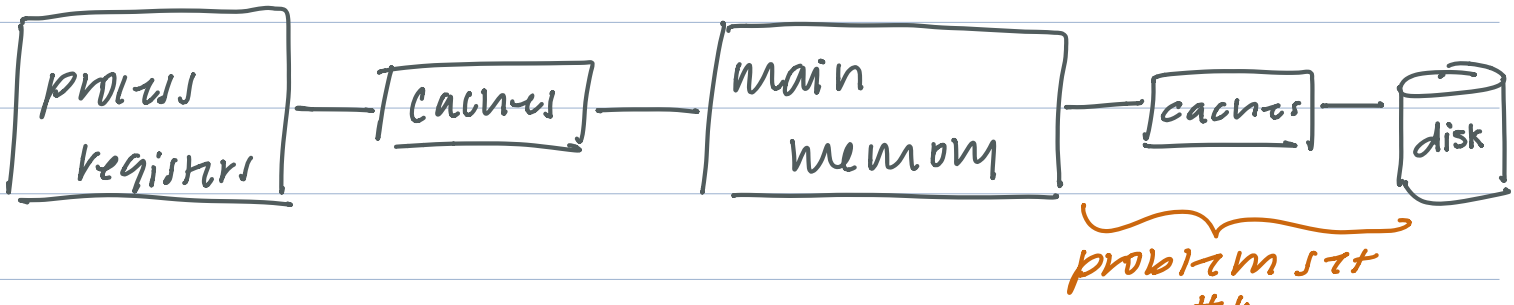
spatial locality

* instructions themselves have temporal and spatial locality

Key: Caching (verb)

using small amount of fast storage to speed up access to some large, slow storage

simple idea, many variants → propels careers



I/O Devices

Unix file ≡ sequence of bytes

↳ treats every I/O device like a file

* some OS have different end-line characters, for example

- files as an abstraction of I/O devices
- virtual memory as an abstraction of main memory
- processes as an abstraction for running programs

```
int main() {
    int fd = STDOUT_FILENO;
    if (isatty(fd)) {
        fd = open(DATAFILE, O_WRONLY | O_CREAT |
                  O_TRUNC | O_SYNC, 0666);
    }
    if (fd < 0) {
        perror("open");
        exit(1);
```

filename; write only; create if doesn't exist; if file exist, throw it out; don't allow prog to cont. until program finishes writing; permissions for everyone to read/write

```
    }
    size_t size = 5120000;
    const char* buf = "6";
    double start = tstamp();
    while(n < size) {
        ssize_t r = write(fd, buf, 1);
        if (r != 1){
            perror("write");
            exit(1);
        }
        n += r;
        if(n % PRINT_FREQUENCY == 0){
            report(n, tstamp() - start);
        }
    }
    close(fd);
    report(n, tstamp()-start);
    fprintf(stderr, "\n");
}
```

5.12 MB

file

write 1 character

buffer w/ data

Original

~55 kb of memory/second

= 0.4 ms / byte

remove O_SYNC flag

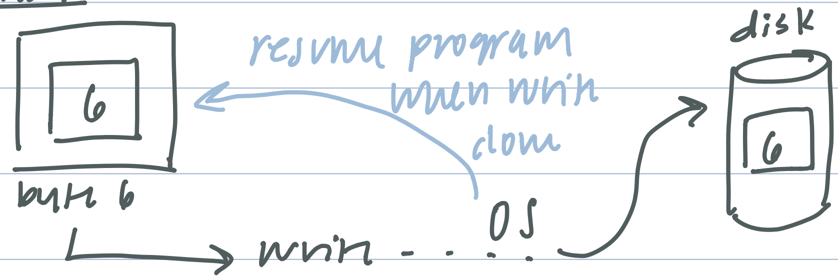
~ 9.2 kb / second

Use stdio (standard i/o library) — fopen, fwrite

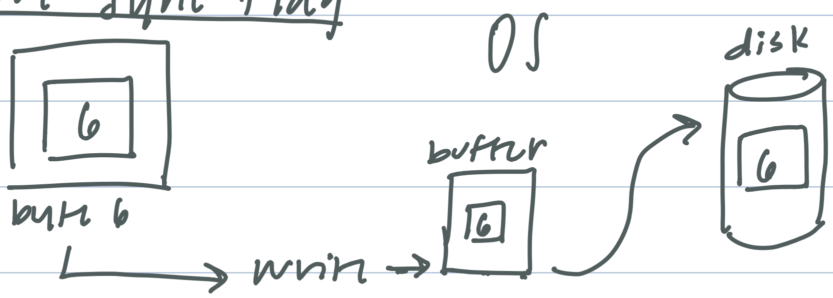
~ 63.2 kb/second

program w0x-y

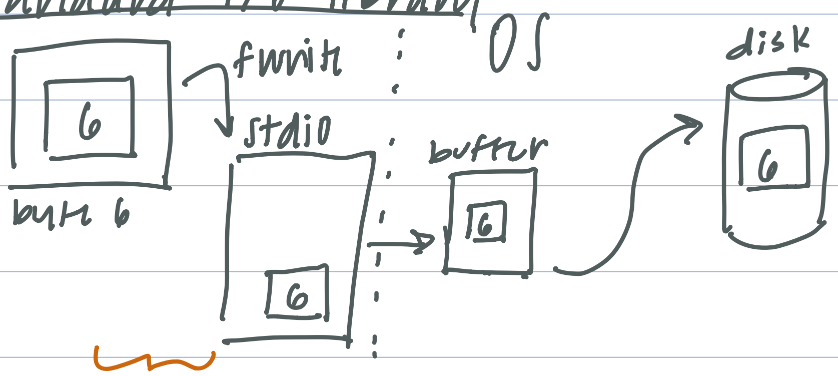
original



remove sync flag



standard i/o library



essentially doing a store