# CS61 Section 5

October 28, 2019

## Strace

- debugs syscalls by another program

strace -o strace.out PROGRAMNAME ARGS...

## Access patterns

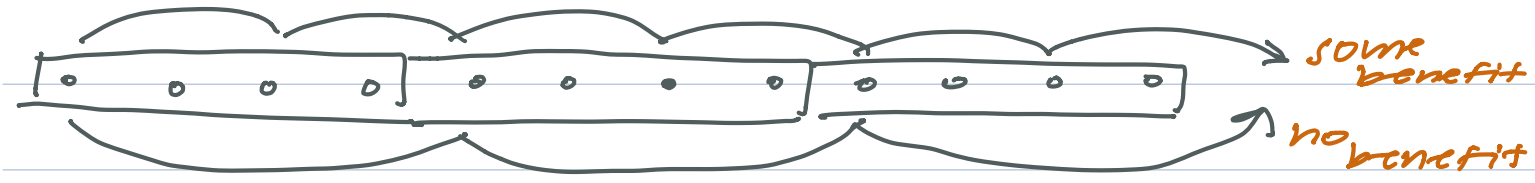
### Sequential access

cache 4 datapoints at the end

### Reverse-sequential access

align to 4-bit groups

### Strided Access




- access data w/ uniform skip
- need large cache

- if we try to read > size of cache, have to make multiple reads on slower storage

strace examples

lseek(0, 4114, SEEK_SET); how to jump to new addresses

strace 05

read(0, "xxxx"..., 4091) = 4091

4091 → actually read # of bytes

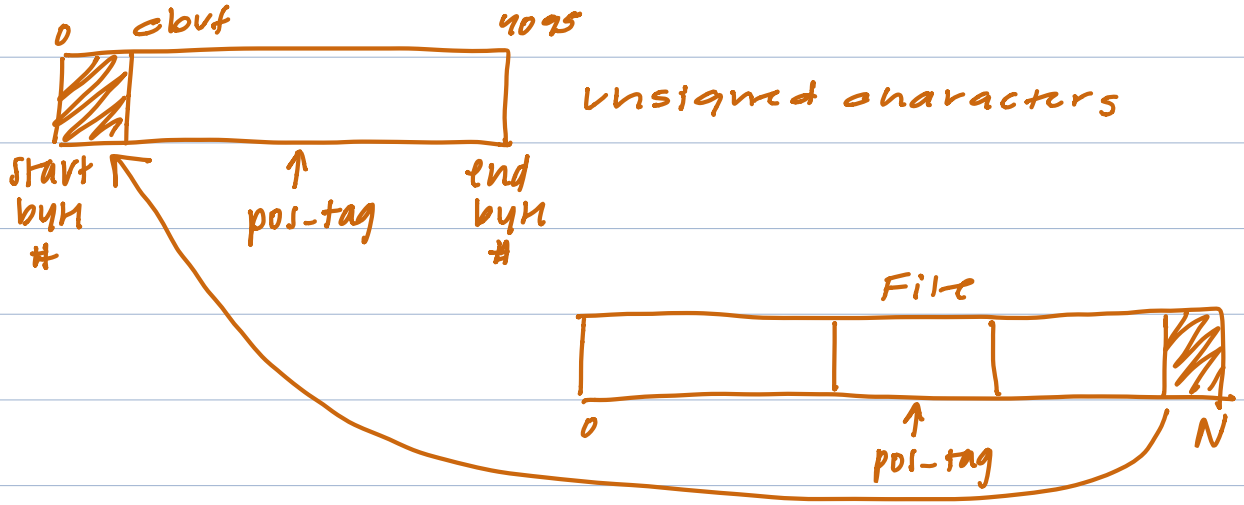
4091 ↓ requested # of bytes to read

vsfd cache

## Single-slot cache

≡ to FILE* from fopen() in standard i/o library

```
struct io61_file {
    int fd;                                  // ← file descriptor
    static constexpr off_t bufsize = 4096; // or whatever
    unsigned char cbuf[bufsize];             // ↳ can change
    off_t tag;      // file offset of first byte in cache (0 when file is opened)
    off_t end_tag;  // file offset one past last valid byte in cache
    off_t pos_tag;  // file offset of next char to read in cache
};
```




Checks

* assuming one byte
  - is the byte inside the buffer or not

keep track of byte # of beginning and end

# Facts about cache representation

- reads or writes but not both
- tag ≤ end_tag
- end_tag − tag ≤ bufsize

  not always equal bc end/start tag can move
- if tag == end_tag

  cache slot is empty (no valid data)
- tag ≤ end_tag and pos_tag ≤ end_tag

| | read | write |
|---|---|---|
| current file pos = | end_tag | tag |

- the next io61_read or io61_write call starts at offset pos_tag.
- if i >= tag and i < end_tag, cbuf[i−tag] = byte at offset i

# Filling cache

```
void io61_fill(io61_file* f) {
    // Fill the read cache with new data, starting from file offset `end_tag`.
    // Only called for read caches.

    // Check invariants.
    assert(f->tag <= f->pos_tag && f->pos_tag <= f->end_tag);
    assert(f->end_tag - f->pos_tag <= f->bufsize);

    /* ANSWER */
    // Reset the cache to empty.
    f->tag = f->pos_tag = f->end_tag;
    // Read data.  read (file descriptor, buffer, buffer size)
    ssize_t n = read(f->fd, f->cbuf, f->bufsize);
    if (n >= 0) {  -> write read
        f->end_tag = f->tag + n;
    }
    -> n is -1 if failed read
    // Recheck invariants (good practice!).
    assert(f->tag <= f->pos_tag && f->pos_tag <= f->end_tag);
    assert(f->end_tag - f->pos_tag <= f->bufsize);
}
```

# Easy read

```
ssize_t io61_read(io61_file* f, char* buf, size_t sz) {
    // Check invariants.
    assert(f->tag <= f->pos_tag && f->pos_tag <= f->end_tag);
    assert(f->end_tag - f->pos_tag <= f->bufsize);

    // The desired data is guaranteed to lie within this cache slot.
    assert(sz <= f->bufsize && f->pos_tag + sz <= f->end_tag);

    /* ANSWER */
    memcpy(buf, &f->cbuf[f->pos_tag - f->tag], sz);
    f->pos_tag += sz;
}
```

assuming hit.

1. calculate index in cbuf and read that index
2. update pos_tag

# Full Read

```
ssize_t io61_read(io61_file* f, char* buf, size_t sz) {
    // Check invariants.
    assert(f->tag <= f->pos_tag && f->pos_tag <= f->end_tag);
    assert(f->end_tag - f->pos_tag <= f->bufsize);

    /* ANSWER */
    size_t pos = 0;
    while (pos < sz) {
        if (f->pos_tag == f->end_tag) {
            io61_fill(f);
            if (f->pos_tag == f->end_tag) {
                break;  or return pos;
            }
        }

        // This would be faster if you used `memcpy`!
        buf[pos] = f->cbuf[f->pos_tag - f->tag];
        ++f->pos_tag;
        ++pos;
    }
    return pos;
}
```

1. loop that reads a byte at a time cbuf-> buf

Easy write

```
ssize_t io61_write(io61_file* f, const char* buf, size_t sz) {
    // Check invariants.
    assert(f->tag <= f->pos_tag && f->pos_tag <= f->end_tag);
    assert(f->end_tag - f->pos_tag <= f->bufsize);

    // Write cache invariant.
    assert(f->pos_tag == f->end_tag);

    // The desired data is guaranteed to lie within this cache slot.
    assert(sz <= f->bufsize && f->pos_tag + sz <= f->tag + f->bufsize);

    /* ANSWER */
    memcpy(&f->cbuf[f->pos_tag - f->tag], buf, sz);
    f->pos_tag += sz;
    f->end_tag += sz;
    return sz;
}
```