Tracing codes TIPS (I)

- Steps
  - Google for a function name
  - Bottom-up backtracking
  - Top-down analyzing
- Tools
  - `grep -nr`: generic text search
  - ctags: semantic-aware search
  - LXR – Linux Cross Reference
    http://lxr.linux.no/#linux+v2.6.32.60/

Readahead flow

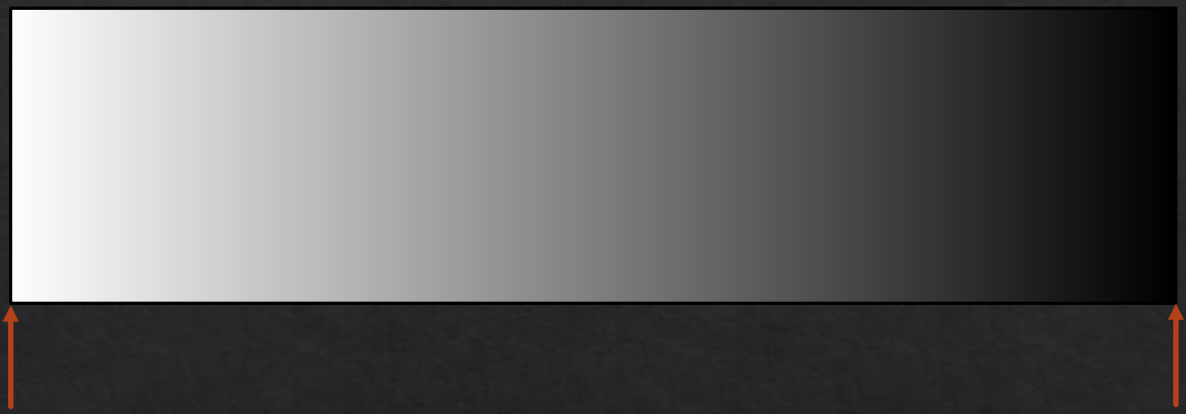
Tracing codes TIPS (II)

- Object Oriented Programming in C

```
/* fs/ext4/file.c */
static const struct vm_operations_struct ext4_file_vm_ops = {
        .fault          = filemap_fault,
        .page_mkwrite   = ext4_page_mkwrite,
};

/* mm/filemap.c */
const struct vm_operations_struct generic_file_vm_ops = {
        .fault          = filemap_fault,
};
```

Readahead algorithm



No readahead
(Pure demand paging)

Full readahead

Better for SSD: reduce unnecessary works

Better for HDD: amortize seek time cost

Readahead algorithm

- How many pages are read?

```
/* mm/filemap.c */
page_cache_async_readahead(mapping, ra, file,page,
                                        offset, ra->ra_pages);
/* mm/readahead.c */
ra->ra_pages = mapping->backing_dev_info->ra_pages;
/* mm/backing-dev.c */
struct backing_dev_info default_backing_dev_info = {
    .ra_pages = VM_MAX_READAHEAD * 1024 / PAGE_CACHE_SIZE,
};
```

Project Goals

- Code reading
  - Part I: How filemap_fault() is set as the page fault handler when mmap() is called
  - Part II: How and when the readahead algorithm takes place when filemap_fault() is invoked
- Revise readahead algorithm
  - Any change that reduces the time between “page fault test program starts !” and “page fault test program ends !”
  - Percentage of reduction varies from machine to machine. Any number larger than 0 is fine. As a reference: 6% on SSD and > 10% on HDD