第 51 組 OS project1

資工二 b02902015 梁智泓 b02902019 陳柏均

```
const struct vm_operations_struct generic_file_vm_ops = {
        .fault          = filemap_fault,
};
```

當發生 Read in file data for page fault 時，便會進到 filemap_fault()這個 function 進 filemap_fault 以後狀況分成兩種

1. Minor page fault：在 page cache 裡面但是作業系統沒有替程式註冊相應的頁面。第二種可能性是工作量太大時將較少使用到的 page 放入空閒頁表，當原程式再度用到的時候將 page 找回來。
2. Major page fault：完全不在 page cache 裡，需要透過 disk IO 去把內容放進 page 裡面。

先用 find_get_cache 先找目標 page,會有兩種結果

```
         */
        page = find_get_page(mapping, offset);
        printk(KERN_CRIT "%s, %X\n", current->comm, (unsigned int)vmf->virtual_address);
        if (likely(page)) {
                /*
                 * We found the page, so try async readahead before
                 * waiting for the lock.
                 */
                //do_async_mmap_readahead(vma, ra, file, page, offset);
        } else {
                /* No page in the page cache at all */
                //do_sync_mmap_readahead(vma, ra, file, offset);
                count_vm_event(PGMAJFAULT);
                mem_cgroup_count_vm_event(vma->vm_mm, PGMAJFAULT);
                ret = VM_FAULT_MAJOR;
retry_find:
                page = find_get_page(mapping, offset);
                if (!page)
                        goto no_cached_page;
        }
```

1. 找到該 page，為 minor page fault。（再根據狀況決定是否 readahead）
2. 該 page 確實沒有在 page cache 裡，更新 ret 值顯示為 VM_FAULT_MAJOR。（再進行同步 mmap_readahead）

此時再度進入 page cache 裡找目標,會有兩種結果

1. 找到該 page，code continue。
2. 沒找到該 page，進入 no_cached_page。

In no_cached_page：

```
no_cached_page:
        /*
         * We're only likely to ever get here if MADV_RANDOM is in
         * effect.
         */
        error = page_cache_read(file, offset);

        /*
         * The page we want has now been added to the page cache.
         * In the unlikely event that someone removed it in the
         * meantime, we'll just come back here and read it again.
         */
        if (error >= 0)
                goto retry_find;

        /*
         * An error return from page_cache_read can result if the
         * system is low on memory, or a problem occurs while trying
         * to schedule I/O.
         */
        if (error == -ENOMEM)
                return VM_FAULT_OOM;
        return VM_FAULT_SIGBUS;
```

page_cache_read()

功能其實就是 alloc 一個新的 page，並把要的內容放到該 page 裡，並連回 page cache，這樣只要在回到 retry_find 便可以找到要的 page

1. return 值大於 0，表示可以正常 alloc，所以回到 retry_find 再跑一次
2. return 值為-ENOMEM，表示現有記憶體太少或是 I/O schedule 有問題。
3. 為其他狀況，直接回傳 VM_FAULT_SIGBUS

最後有個 error checking 的東西是 page_not_uptodate：

```
page_not_uptodate:
        /*
         * Umm, take care of errors if the page isn't up-to-date.
         * Try to re-read it _once_. We do this synchronously,
         * because there really aren't any performance issues here
         * and we need to check for errors.
         */
        ClearPageError(page);
        error = mapping->a_ops->readpage(file, page);
        if (!error) {
                wait_on_page_locked(page);
                if (!PageUptodate(page))
                        error = -EIO;
        }
        page_cache_release(page);

        if (!error || error == AOP_TRUNCATED_PAGE)
                goto retry_find;

        /* Things didn't work out. Return zero to tell the mm layer so. */
        shrink_readahead_size_eio(file, ra);
        return VM_FAULT_SIGBUS;
}
```

Pure demand paging v.s. readahead algorithm

Pure Demand Paging：

```
# of major pagefault: 6566
# of minor pagefault: 229
# of resident set size: 26572 KB
```

Readahead：

```
[  120.958042]  page fault test program ends !
# of major pagefault: 1256
# of minor pagefault: 5499
# of resident set size: 26648 KB
bearman@OS2015:~$ _
```

從上面的結果可以得知，Pure Dmand Paging 讓 major pagefault 大量的增加。換言之，pagefault 的 loading 也會大幅度的上升，因為 Disk I/O 相當 time consuming。因為他並不會先 readahead 以降低 loading，而是 program 要多少，就給他多少。

而 Readahead Algorithm 因為會先將部分 program 沒要的，先 load 進 page cache 中，以增加 minor pagefault 減少 major pagefault，讓 loading 下降