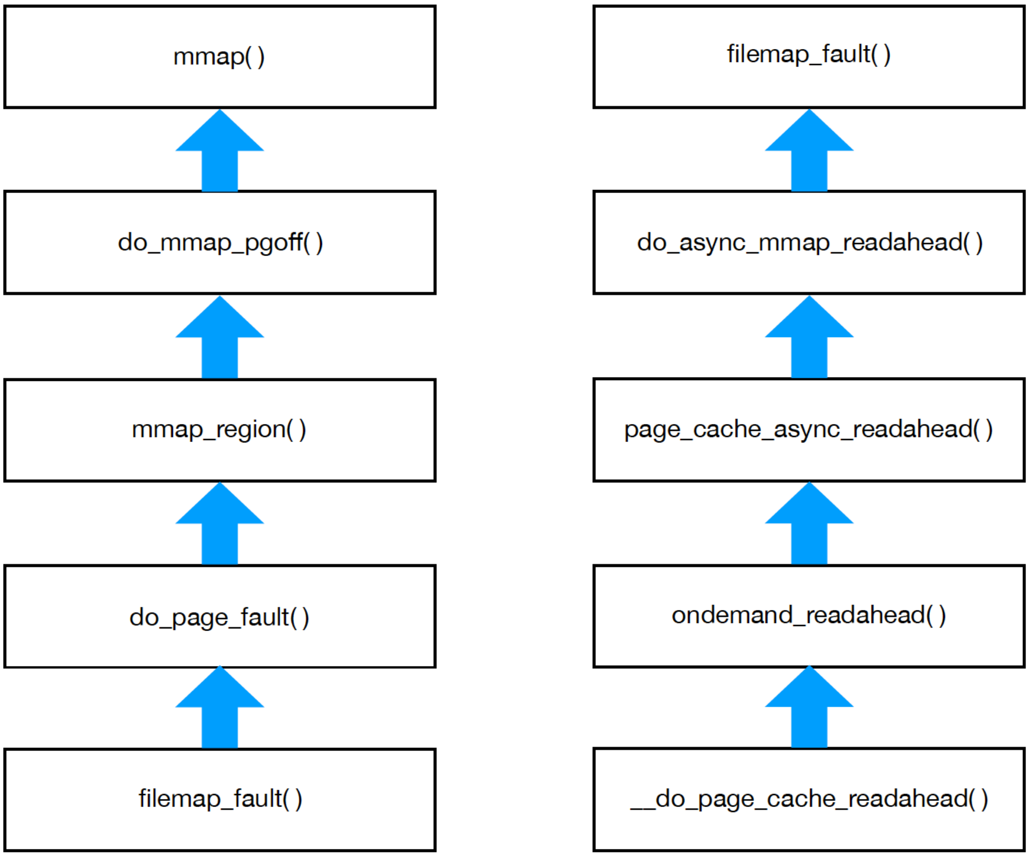
Operating System Project 3

Team53: b05902121 黃冠博 b05902019 蔡青邑

June 20, 2018

Code Reading

Readahead Flow



Part I

How `filemap_fault()` is set as the page fault handler when `mmap()` is called?

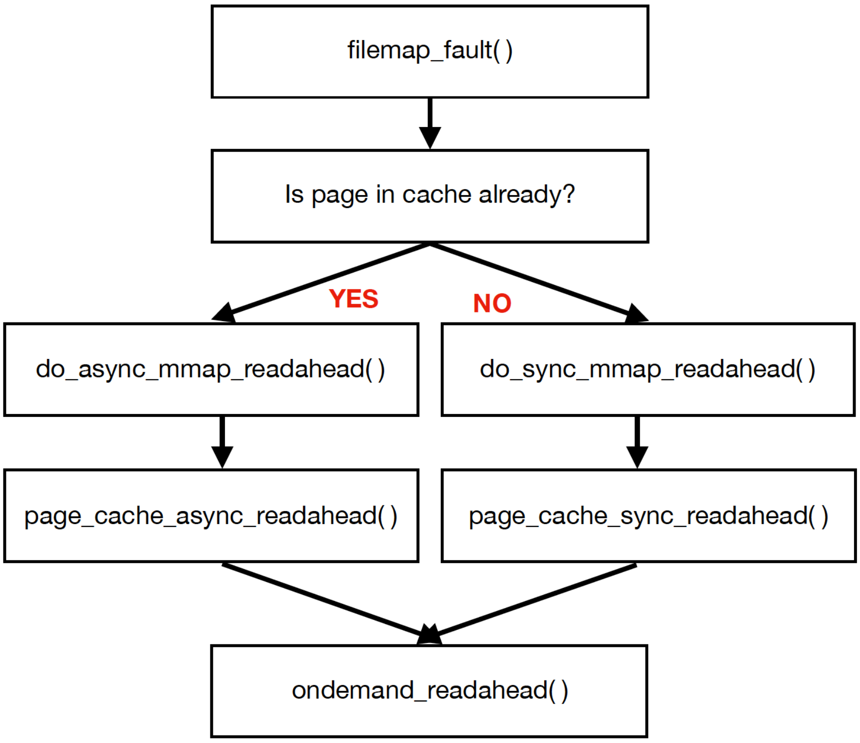
mmap()被呼叫的時候，會去 kernel 呼叫 do_mmap_pgoff( )，裡面有個函式 get_unmapped_area( )，他會取得要 map 的 address 然後確定是 valid 的。
另外，vm_flags = calc_vm_prot_bits(prot)| calc_vm_flag_bits(flags)......，會把傳進去的 flag 都轉成 vm_flags
最後面 return 的地方，呼叫了 mmap_region()，他會做一個 struct vm_area_struct，這個是要 map 到的 address space 區域的資訊(包含了 mapping 要用到的 address, length, offset, VM flags)
error = file->f_op->mmap(file, vma)，對應的 filesystem 會被設定，舉例來說，當一個 file 被存在 disk 上的時候，會用ext4_file_mmap()來實作。這個過程中就會用到 generic_file_mmap，裡面有一行vma->vm_ops =&generic_file_vm_ops，如下圖，struct vm_operations_struct 其中包括了.fault = filemap_fault的初始化

動作。

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

Part II

How and when the readahead algorithm takes place when `filemap_fault` is invoked?



`filemap_fault` is invoked via the vma operations vector for a mapped memory region to read in file data during a page fault. **It checks whether the required page is already in cache.**
If the required page is already in cache, it calls the function `do_async_mmap_readahead( )`, inside it is a function `page_cache_async_readahead( )`, which loads pages into memory asynchronously. Asynchronous in this situation means that it doesn't block after a request. If readahead is not needed, `if (!ra->ra_pages)` returns immediately. Before running readahead, it needs to check whether there is I/O congestion. If yes, readahead has to be deferred. Finally, the `ondemand_readahead( )` function implements the readahead algorithm.
If the required page isn't in cache, it runs the function `do_sync_mmap_readahead( )`, inside it is the function `page_cache_sync_readahead( )` and leads to the function `ondemand_readahead( )`, too. There are some main parts in this function.

- If `offset == 0`, it means that it is the starting of the file. Then `goto initial_readahead`, `get_init_ra_size ( )` sets the initial window.
- If it isn't the start of the file, check if it is a subsequent sequential access. If yes, ramp up the window.
- If it is oversize read or sequential cache miss, `goto initial_readahead`, `get_init_ra_size( )` sets the initial window.
- The above scenarios eventually end up in the function `ra_submit()` which includes the function `__do_page_cache_readahead( )`.

- If it isn't the scenarios above, it is a standalone, small random read. Read as is, using the function `__do_page_cache_readahead( )`, and do not pollute the readahead state.

```
1  __do_page_cache_readahead(struct address_space *mapping, struct file *filp, pgoff_t
       offset, unsigned long nr_to_read, unsigned long lookahead_size)
2  {    ......
3       for (page_idx = 0; page_idx < nr_to_read; page_idx++) {
4           pgoff_t page_offset = offset + page_idx;
5           if (page_offset > end_index) break;
6           rcu_read_lock();
7           page = radix_tree_lookup(&mapping->page_tree, page_offset);
8           rcu_read_unlock();
9           if (page) continue;
10          page = page_cache_alloc_cold(mapping);
11          if (!page) break;
12          page->index = page_offset;
13          list_add(&page->lru, &page_pool);
14          if (page_idx == nr_to_read - lookahead_size) SetPageReadahead(page);
15          ret++;
16      }
17      if (ret) read_pages(mapping, filp, &page_pool, ret);
18 }
```

The for loop preallocates as many pages as needed. During the readahead progress, some pages may have been loaded into the memory already. Line 6 to 8 checks whether the pages are in cache yet. If no, `page_cache_alloc_cold( )` allocates memory pages and adds the pages into the page pool. After the loop is finished, `read_pages( )` implements the I/O operations and reads the files from disk.

Revise readahead algorithm

VM_MAX_READAHEAD = 16

```
gerber@gerber-VirtualBox:~/hw3$ dmesg | grep 'page fault'
[63288.037532]  page fault test program starts !
[63293.074351]  page fault test program ends !
```

response time(sec): 5s

VM_MAX_READAHEAD = 128(default)

```
# of major pagefault: 4158
# of minor pagefault: 2640
# of resident set size: 26604 KB
gerber@gerber-VirtualBox:~/hw3$ dmesg | grep 'page fault'
[67314.857973]  page fault test program starts !
[67316.511866]  page fault test program ends !
gerber@gerber-VirtualBox:~/hw3$ sudo blockdev --getra /dev/sda
256
```

response time(sec): 1.7s

VM_MAX_READAHEAD = 512

```
/* readahead.c */
#define VM_MAX_READAHEAD    512 /* kbytes */
#define VM_MIN_READAHEAD    16  /* kbytes (includes current page) */
```

```
# of major pagefault: 352
# of minor pagefault: 6445
# of resident set size: 26680 KB
gerber@gerber-VirtualBox:~/hw3$ dmesg | grep 'page fault'
[   80.860639]  page fault test program starts !
[   81.560800]  page fault test program ends !
gerber@gerber-VirtualBox:~/hw3$ sudo blockdev --getra /dev/sda
1024
```

response time(sec): 0.7s

VM_MAX_READAHEAD = 1024

```
/* readahead.c */
#define VM_MAX_READAHEAD    1024    /* kbytes */
#define VM_MIN_READAHEAD    16  /* kbytes (includes current page) */
```

```
# of major pagefault: 185
# of minor pagefault: 6611
# of resident set size: 26676 KB
gerber@gerber-VirtualBox:~/hw3$ dmesg | grep 'pagefault'
gerber@gerber-VirtualBox:~/hw3$ dmesg | grep 'page fault'
[   44.803792]  page fault test program starts !
[   45.316353]  page fault test program ends !
gerber@gerber-VirtualBox:~/hw3$ sudo blockdev --getra /dev/sda
2048
```

response time(sec): 0.5s

When VM_MAX_READAHEAD is larger, the number of major pagefaults decrease. Hence, the running time becomes shorter.
However, I also tried VM_MAX_READAHEAD = 2048, but the response time became even longer than the default. I think it is because there are too many minor pagefaults. So the whole thing is actually about balancing major and minor pagefaults.

Bonus

How to improve throughput on disk I/O?

- 擴大 virtual machine 的 memory 大小，這樣 buffer cache 的大小也可以增加
- 減少 major page fault 的數量

實作：
使用 command line 指令複製 1GB 的檔案並測速：`sudo dd if=/dev/zero of=/testfile bs=51200 count =20000 oflag=direct`

VM_MAX_READAHEAD = 256

```
gerber@gerber-VirtualBox:~/hw3$ sudo dd if=/dev/zero of=/testfile bs=51200 count
=20000 oflag=direct
20000+0 records in
20000+0 records out
1024000000 bytes (1.0 GB) copied, 5.35004 s, 191 MB/s
```

Average running speed about 200 MBs.

VM_MAX_READAHEAD = 512

```
gerber@gerber-VirtualBox:~/hw3$ sudo dd if=/dev/zero of=/testfile bs=51200 count
=20000 oflag=direct
20000+0 records in
20000+0 records out
1024000000 bytes (1.0 GB) copied, 3.49135 s, 293 MB/s
gerber@gerber-VirtualBox:~/hw3$ sudo dd if=/dev/zero of=/testfile bs=51200 count
=20000 oflag=direct
20000+0 records in
20000+0 records out
1024000000 bytes (1.0 GB) copied, 4.79841 s, 213 MB/s
gerber@gerber-VirtualBox:~/hw3$ sudo dd if=/dev/zero of=/testfile bs=51200 count
=20000 oflag=direct
20000+0 records in
20000+0 records out
1024000000 bytes (1.0 GB) copied, 4.00136 s, 256 MB/s
gerber@gerber-VirtualBox:~/hw3$ sudo dd if=/dev/zero of=/testfile bs=51200 count
=20000 oflag=direct
20000+0 records in
20000+0 records out
1024000000 bytes (1.0 GB) copied, 3.88859 s, 263 MB/s
```

Average running speed about 255 MBs.