Project 3 Report

Team 23

402125039 葉家瑋

404420020 張庭瑋

Implementation and Result

在檔案 linux-2.6.38/mm/filemap.c 中，註解以下兩行：

```
1560 //do_async_mmap_readahead(vma, ra, file, page, offset);
1563 //do_sync_mmap_readahead(vma, ra, file, offset);
```

即能得到以下結果：

```
# of major pagefault: 6567
# of minor pagefault: 228
# of resident set size: 26604 KB
```

Case Study

左圖為 readahead，右圖為 pure demand paging。

```
# of major pagefault: 1291
# of minor pagefault: 5508
# of resident set size: 26672 KB

real    0m1.521s
user    0m0.008s
sys     0m0.156s
```

```
# of major pagefault: 6567
# of minor pagefault: 228
# of resident set size: 26604 KB

real    0m3.407s
user    0m0.032s
sys     0m0.528s
```

分析兩者的 major page fault 與 minor page fault，發現兩者的 page fault 總和約略相同，但 pure demand paging 有極高的比率是 major page fault，可以推斷 readahead 大幅減少了 major page fault 的次數。

再比較兩者的執行時間，readahead 明顯少於 pure demand paging，顯示 major page fault 才是影響執行效率的關鍵。

Trace Code

此函數為預設的 mmap：

```
1662 int generic_file_mmap(struct file * file, struct vm_area_struct *
     vma)
1663 {
1664     struct address_space *mapping = file->f_mapping;
1665
1666     if (!mapping->a_ops->readpage)
1667         return -ENOEXEC;
1668     file_accessed(file);
1669     vma->vm_ops = &generic_file_vm_ops;
```

```
1670     vma->vm_flags |= VM_CAN_NONLINEAR;
1671     return 0;
1672 }
```

其中，vma 為一 Memory Region 物件。1669 行中，此函數將 vma 的 operation struct 指定為 &generic_file_vm_ops，這個物件在上一段宣告：

```
1656 const struct vm_operations_struct generic_file_vm_ops = {
1657     .fault      = filemap_fault,
1658 };
```

它將 page fault 的處理方法指定為 filemap_fault，即本次作業需要修改的函數。

在 filemap_fault()中，檢查造成 page fault 的 vma 是否以 cache 的形式存在記憶體中，若存在，則執行非同步 readahead：

```
1560         do_async_mmap_readahead(vma, ra, file, page, offset);
```

若不存在，即為 major fault，執行同步 readahead，試著把 page 寫入 cache：

```
1563         do_sync_mmap_readahead(vma, ra, file, offset);
```