Operating System: Project 3

資工二 B04902051 林承豫
資工二 B04902053 鄭淵仁

Part 1: Code Reading

- **How readahead is called when page faults occur?**

The mmap() library call converts the offset in bytes to an offset in pages, then calls the mmap_pgoff() system call.

The mmap_pgoff() system call fetches the struct file * corresponding to the file descriptor argument, and calls do_mmap_pgoff().

do_mmap_pgoff() calculates the actual address and length that will be used based on the hint and the available address space, converts the provided flags into VM flags, and tests for permission to perform the mapping.It then calls mmap_region().

```
unsigned long do_mmap_pgoff(struct file *file, unsigned long addr,
                        unsigned long len, unsigned long prot,
                        unsigned long flags, unsigned long pgoff)
```

```
return mmap_region(file, addr, len, flags, vm_flags, pgoff);
```

mmap_region() removes any prior mappings in the area being replaced by the new mapping, performs memory accounting and creates the new struct vm_area_struct describing the region of the address space being mapped. It then calls the file's ->mmap() implementation，for an ordinary file on ext4, ext4_file_mmap() is used.

```
error = file->f_op->mmap(file, vma);
```

```
static const struct vm_operations_struct ext4_file_vm_ops = {
        .fault          = filemap_fault,
        .page_mkwrite   = ext4_page_mkwrite,
};

static int ext4_file_mmap(struct file *file, struct vm_area_struct *vma)
{
        struct address_space *mapping = file->f_mapping;

        if (!mapping->a_ops->readpage)
                return -ENOEXEC;
        file_accessed(file);
        vma->vm_ops = &ext4_file_vm_ops;
        vma->vm_flags |= VM_CAN_NONLINEAR;
        return 0;
}
```

ext4_file_mmap() would change vma->vm_ops to ext4_file_vm_ops, and .fault in ext4_file_vm_ops would be assign to filemap_fault.

- **Implementation of readahead algorithm**

  當有 page fault 產生時，linux 的作法流程如下：

1. filemap_fault()會執行 do_async_mmap_readahead()
2. do_async_mmap_readahead()會執行

```
page_cache_async_readahead(mapping, ra, file, page, offset, ra->ra_pages)
```

   - 其中在 mm/readahead.c 把 ra->ra_pages 定為：

```
ra->ra_pages = mapping->backing_dev_info->ra_pages;
```

   - 而 backing_dev_info 又定義在 mm/backing-dev.c：

```
struct backing_dev_info default_backing_dev_info = {
        .name           = "default",
        .ra_pages       = VM_MAX_READAHEAD * 1024 / PAGE_CACHE_SIZE,
        .state          = 0,
        .capabilities   = BDI_CAP_MAP_COPY,
        .unplug_io_fn   = default_unplug_io_fn,
};
```

   - 其中的 VM_MAX_READAHEAD 是定義在 include/linux/mm.h：

```
#define VM_MAX_READAHEAD        128     /* kbytes */
#define VM_MIN_READAHEAD        16      /* kbytes (includes current page) */
```

3. page_cache_async_readahead()會用 req_size 來接上述的 ra->ra_pages，然後執行

```
ondemand_readahead(mapping, ra, filp, true, offset, req_size)
```

4. ondemand_readahead()就會呼叫

```
__do_page_cache_readahead(mapping, filp, offset, req_size, 0)
```

   來預讀檔案內容

所以，可以透過更改VM_MAX_READAHEAD的數值來改變reg_size，這樣就可以手動設定一次要預讀多少個page。

Part 2: Revise the readahead algorithm for smaller response time

- **Experiments**

從上一題read code中，我們發現Linux會依據VM_MAX_READAHEAD的值來計算page的數量，因此我們就去直接去改動VM_MAX_READAHEAD在“include/linux/mm.h” define的值，來試試看是否會對page fault數量以及時間有影響。

我們實驗用的電腦是I5-4200、HDD 7200轉，在Virtual Box上裝32位元的Ubuntu，記憶體給3G。

另外，為了降低Linux其他運作的程式造成實驗的誤差，我們在每組數據上都實驗了5次。最後的實驗結果如下：

（Linux預設的VM_MAX_READAHEAD是128）

表一 不同VM_MAX_READAHEAD的response time (秒)

	1	128	256	512
exp 1	34.250	30.334	22.002	**4.979**
exp 2	30.122	26.079	21.500	**5.196**
exp 3	29.470	25.611	21.494	**4.605**
exp 4	29.863	25.974	21.705	**5.035**
exp 5	29.537	28.690	21.051	**4.658**
平均	30.6484	27.5376	21.5504	**4.8946**

表二 不同VM_MAX_READAHEAD的major pagefault

	1	128	256	512
exp 1	6574	4203	2027	**356**
exp 2	6581	4203	2027	**356**
exp 3	6581	4203	2027	**356**
exp 4	6581	4203	2027	**356**
exp 5	6581	4203	2027	**356**
平均	6579.6	4203.0	2027.0	**356.0**

- **Discussions**

從**表一**可以觀察到：把VM_MAX_READAHEAD設成512之後，response time特別短。可以推測預讀的page數量越多，讀大型檔案的速度越快。

從**表二**可以觀察到：把VM_MAX_READAHEAD設成512之後，major pagefault變得特別少。而這是因為預讀的page數量越多，pagefault會越少。

而pagefault越少，response time自然就會比較短。這可以從**表一**和**表一**數據驗證。

References

1. Connection between mmap user call to mmap kernel call：https://stackoverflow.com/questions/9798008/connection-between-mmap-user-call-to-mmap-kernel-call
2. Linux 的預讀：http://m.blog.chinaunix.net/uid-30126070-id-5157819.html
3. Free electrons - Linux：http://elixir.free-electrons.com/linux/v2.6.32.60/source