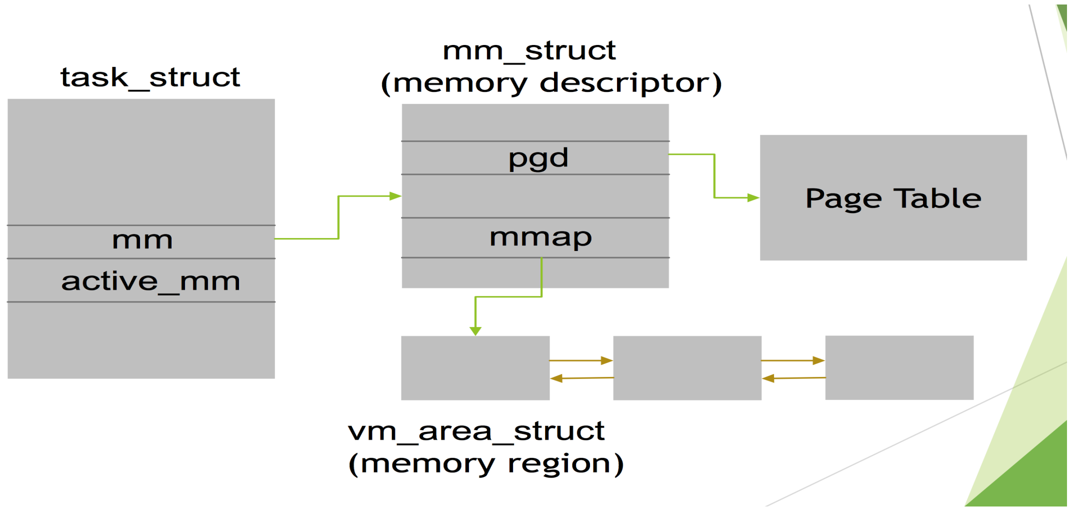
OS Project 3

Team 35

資工三 B03902125 林映廷

資工三 B03902129 陳鵬宇

Trace mmap()



In ***linux/sched.h*** , ***task_struct*** of each process contains ***mm_struct*** in ***linux/mm_types.h***. ***mm_struct*** is memory descriptor, containig informations about the utilization of memory. Its first member is ***vm_area_struct***, or memory region, in ***linux/mm_types.h*** . ***vm_area_struct*** is a linked list of virtual memory area(VMA) .

In addition, for file-backed memory regions, the operation is implemented in ***linux/filemap.c***

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

Hence, it will invoke ***filemap_fault()*** when a page fault occurs.

Trace filemap_fault()

When a page fault occurs, the invoked ***filemap_fault()*** will check whether the required page is in the page cache by ***find_get_fault()*** at first.

If we find the page in page cache, we will try to call ***do_async_mmap_readahead()***; otherwise, we will try to call ***do_sync_mmap_readahead()***.

We will find the page by calling ***do_async_mmap_readahead()*** and ***do_sync_mmap_readahead()*** if ***MADV_RANDOM*** isn't in effect. If we find the page by ***find_get_page()***, we will lock the page and check whether it is truncated and up-to-date. After checking its size under page lock, we return the required page.

However, if ***MADV_RANDOM*** is in effect, we *goto* ***no_cached_page*** . ***no_cached_page*** will simply do ***page_cache_read()*** to read the required page and go back to do ***find_get_page()*** again.

```
no_cached_page:
    /*
     * We're only likely to ever get here if MADV_RANDOM is in
     * effect.
     */
```

Of course, during readahead algorithm, ***filemap_fault()*** will call ***do_aync_mmap_readahead()***, ***page_cache_async_readahead()***, ***ondemand_readahead()***, and ***_do_page_cache_readahead()*** to page those marked in the page cache for readahead.

Implementation

By tracing ***filemap_fault()*** , we find two methods to do pure demand paging:

(1) We return immediately in the beginning of ***do_async_mmap_readahead()*** and ***do_sync_mmap_readahead()*** .

```
static void do_async_mmap_readahead(struct vm_area_struct *vma,
                    struct file_ra_state *ra,
                    struct file *file,
                    struct page *page,
                    pgoff_t offset)
{
    return;
```

```
static void do_sync_mmap_readahead(struct vm_area_struct *vma,
                   struct file_ra_state *ra,
                   struct file *file,
                   pgoff_t offset)
{
    return;
```

(2) We also comment on the ***do_async_mmap_readahead()*** and ***do_sync_mmap_readahead()*** in ***filemap_fault()*** .

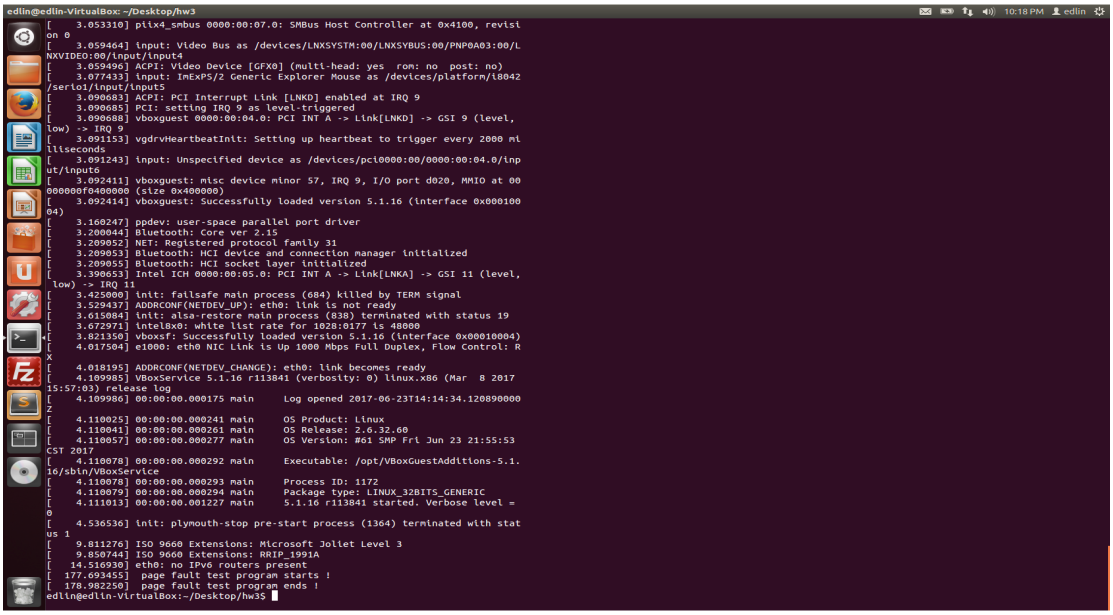
```
        page = find_get_page(mapping, offset);
        if (likely(page)) {
            /*
             * We found the page, so try async readahead before
             * waiting for the lock.
             */
            //do_async_mmap_readahead(vma, ra, file, page, offset);
            lock_page(page);

            /* Did it get truncated? */
            if (unlikely(page->mapping != mapping)) {
                unlock_page(page);
                put_page(page);
                goto no_cached_page;
            }
        } else {
            /* No page in the page cache at all */
            //do_sync_mmap_readahead(vma, ra, file, offset);
            count_vm_event(PGMAJFAULT);
            ret = VM_FAULT_MAJOR;
    retry_find:
            page = find_lock_page(mapping, offset);
            if (!page)
```

Pure Demand Paging vs. Readahead Algorithm

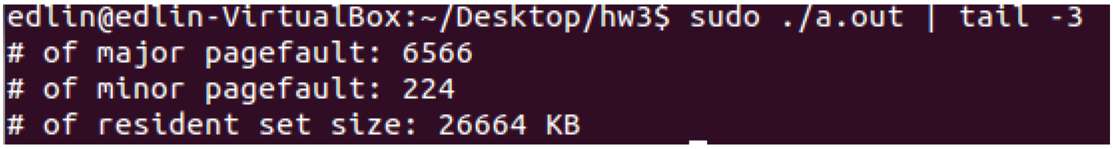
(1) ***Readahead Algorithm***

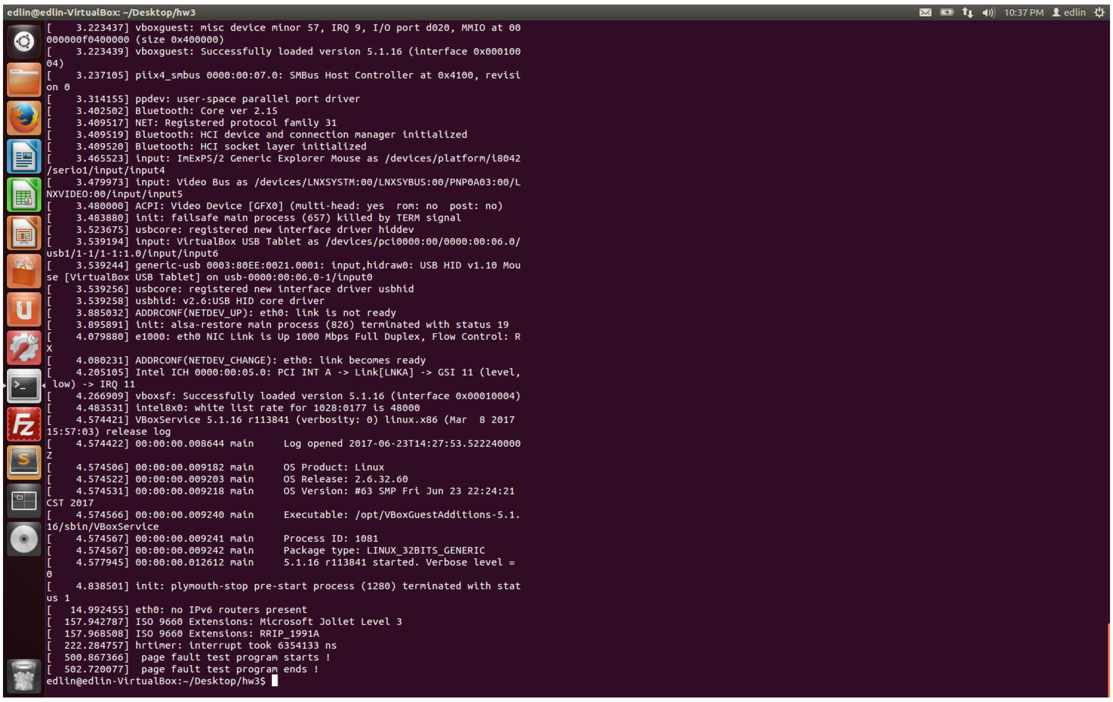
```
edlin@edlin-VirtualBox:~/Desktop/hw3$ sudo ./a.out | tail -3
# of major pagefault: 4200
# of minor pagefault: 2591
# of resident set size: 26664 KB
```



In ***Readahead Algorithm***, the pager pre-load more pages. If it guesses correctly on sequential I/O, it will improve the performance; otherwise, it make redundant overhead. Moreover, Its number of major pagefault is more than ***Pure Demand Paging***'s one.

(2) Pure Demand Paging





In ***Pure Demand Paging***, it doesn't pre-load more pages, in contrast to ***Readahead Algorithm***. That is, it cannot optimize on sequential I/O. Moreover, Its number of minor pagefault is less than ***Readahead Algorithm***'s one.