Operating System, Spring 2018

Project 3

DUE DATE: June 20, 2018

Team 16: B03505053 曾彥青 B03902129 陳鵬宇

Code reading

- How readahead is called when page faults occur?
  - mmap()
  - filemap_fault()

The structure **task_struct** (defined in "linux/sched.h") contains a substructure **mm_struct** (defined in "linux/mm_types.h") which is the memory descriptor, storing some useful information about the usage of memory.

In **mm_struct**, the first member is **vm_area_struct**, the *memory region* (defined in "linux/mm_types.h") which is a linked list of virtual memory area.

Thus we can get the following diagram:

task_struct → mm_struct → vm_area_struct → fault

In "linux/filemap.c", we can find following operations structure:

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

Therefore, when a page fault occurs, it'll consequently invoke **filemap_fault()**, which will then check whether the required page is in the page cache by function **find_get_page()** first.
There are two conditions:

(a) The page is in the page cache.

(b) The page is not in the page cache.

For (a), we'll execute **async_readahead** to read pages.
For (b), we'll execute **sync_readahead** to read the required page and *readahead* other pages to cache.

Both **do_async_mmap_readahead()** and **do_sync_mmap_readahead()** will check whether VMA is randomly reading by **VM_RandomReadHint()**.
If **VM_RandomReadHint()** returns true, it's no need to do *readahead*, therefore both functions will return; otherwise, both functions will keep executing.
Finally, we'll find the page by **async_readahead** and **sync_readahead** if **MADV_RANDOM** has no effect (VMA isn't reading randomly). If we found the page (checking by **find_get_page()**), we'll 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.
If **MARD_RANDOM** has no effect, we goto **no_cached_page**, and it'll execute **page_cache_read()** to read the required page and go back to **find_get_page()**.

Revise the readahead algorithm for smaller response time

We modify the original readahead function in "mm/readahead.c" and find that **struct file_ra_state** defined in "include/linux/fs.h" controls the readahead state of the file.
We also change **get_next_ra_size()** to decide the size of readahead.

```
static unsigned long get_next_ra_size(struct file_ra_state *ra, unsigned long max) {
    unsigned long cur = ra->size;
    unsigned long newsize;

    int ORIGINAL = false;

    if (ORIGINAL) {                     // original algorithm
        if (cur < max / 16)
            newsize = 4 * cur;
        else
            newsize = 2 * cur;
    } else {                            // revised algorithm
        if (cur < max / 32)
            newsize = 16 * cur;
        else if (cur < max / 16)
            newsize = 8 * cur;
        else
            newsize = 4 * cur;
    }

    return min(newsize, max);
}
```

There are two cases, and we test each case for 5 times:

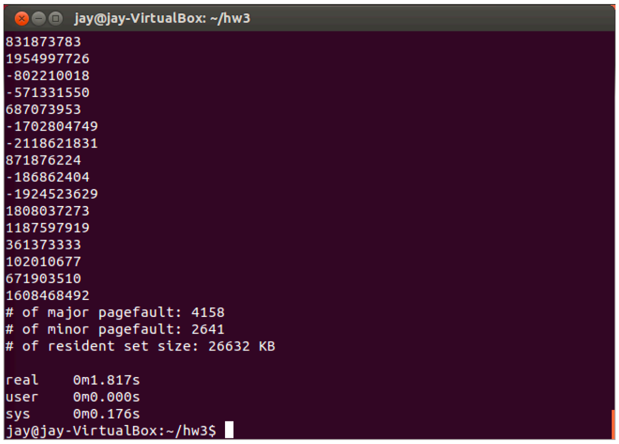
1. Original readahead.c

	1	2	3	4	5	*average*
# of major pagefault	4158	4158	4158	4158	4158	4158
# of minor pagefault	2639	2641	2640	2640	2641	2640.2
# of resident set size	26620	26636	26624	26632	26632	26628.8
real(*sec*)	1.845	1.818	1.833	1.904	1.817	1.8434
user(*sec*)	0	0.036	0	0.016	0	0.0104
sys(*sec*)	0.018	0.16	0.196	0.0192	0.176	0.11384

2. Revised readahead.c

	1	2	3	4	5	*average*
# of major pagefault	178	178	178	178	178	178
# of minor pagefault	6618	6618	6619	6620	6620	6619
# of resident set size	26476	26476	26476	26476	26476	26476
real(*sec*)	0.35	0.371	0.378	0.359	0.354	0.3624
user(*sec*)	0	0	0.012	0.012	0.004	0.0056
sys(*sec*)	0.064	0.064	0.06	0.06	0.068	0.0632

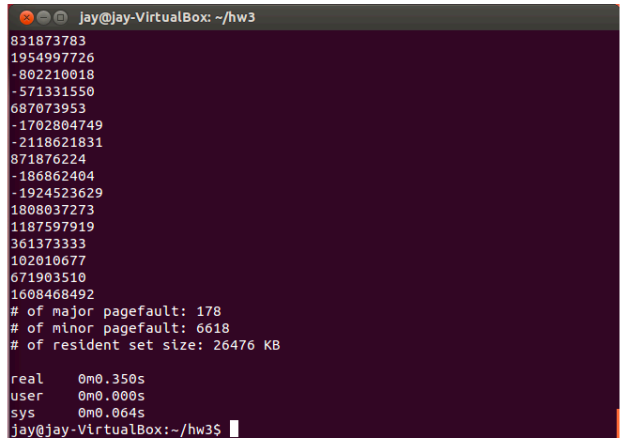
Screenshots



```
jay@jay-VirtualBox: ~/hw3
831873783
1954997726
-802210018
-571331550
687073953
-1702804749
-2118621831
871876224
-186862404
-1924523629
1808037273
1187597919
361373333
102010677
671903510
1608468492
# of major pagefault: 4158
# of minor pagefault: 2641
# of resident set size: 26632 KB

real    0m1.817s
user    0m0.000s
sys     0m0.176s
jay@jay-VirtualBox:~/hw3$
```

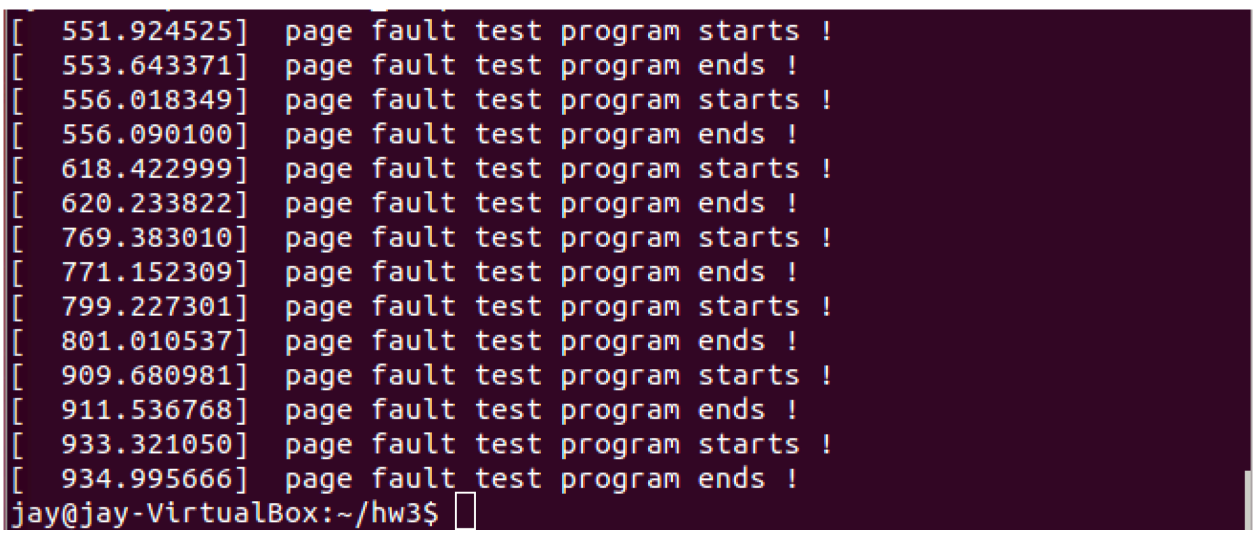
(a) Original algorithm



```
jay@jay-VirtualBox: ~/hw3
831873783
1954997726
-802210018
-571331550
687073953
-1702804749
-2118621831
871876224
-186862404
-1924523629
1808037273
1187597919
361373333
102010677
671903510
1608468492
# of major pagefault: 178
# of minor pagefault: 6618
# of resident set size: 26476 KB

real    0m0.350s
user    0m0.000s
sys     0m0.064s
jay@jay-VirtualBox:~/hw3$
```

(b) Revised algorithm



```
[  551.924525]  page fault test program starts !
[  553.643371]  page fault test program ends !
[  556.018349]  page fault test program starts !
[  556.090100]  page fault test program ends !
[  618.422999]  page fault test program starts !
[  620.233822]  page fault test program ends !
[  769.383010]  page fault test program starts !
[  771.152309]  page fault test program ends !
[  799.227301]  page fault test program starts !
[  801.010537]  page fault test program ends !
[  909.680981]  page fault test program starts !
[  911.536768]  page fault test program ends !
[  933.321050]  page fault test program starts !
[  934.995666]  page fault test program ends !
jay@jay-VirtualBox:~/hw3$
```

Bonus

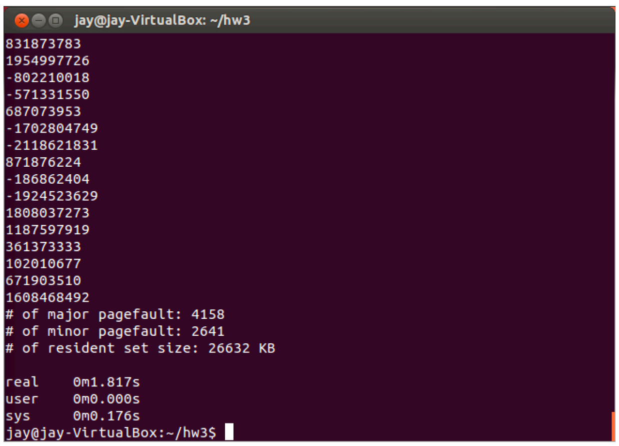
We use the following command to change the size of **VM_MAX_READHEAD** and test the revised algorithm, and we don't revise any code.

```
$ sudo /sbin/blockdev --setra 512 /dev/sda
$ sudo /sbin/blockdev --setra 2048 /dev/sda
$ sudo /sbin/blockdev --setra 8192 /dev/sda
```

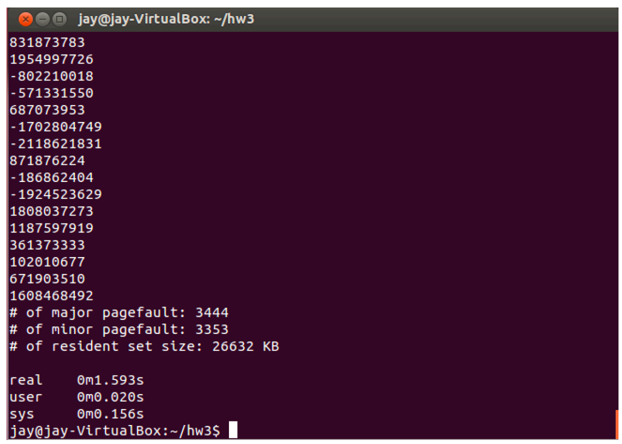
Different size of **VM_MAX_READHEAD**:

	128(original)	512	2048	8192
average time(*sec*)	1.8434	1.593	0.3624	0.281

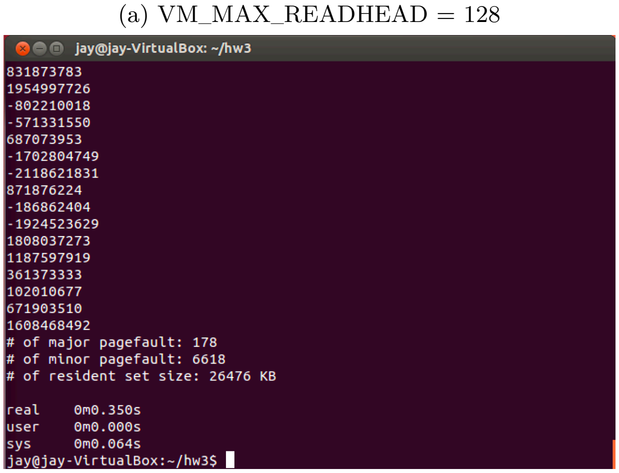
Screenshots



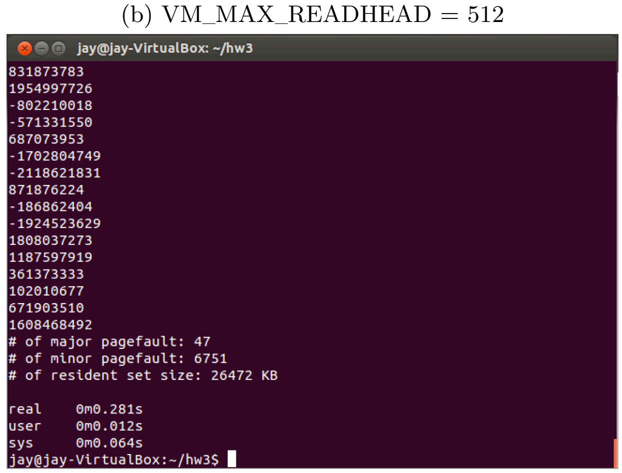
(a) VM_MAX_READHEAD = 128



(b) VM_MAX_READHEAD = 512



(c) VM_MAX_READHEAD = 2048



(d) VM_MAX_READHEAD = 8192

It's obvious that after enlarging **VM_MAX_READHEAD**, the real time is reduced. The reason is that when the parameter is enlarged, it'll make the number of pages we can pre-read each time more, thus letting the times of major page fault lesser, and improving the efficiency.