OS project3-Read Ahead Algorithm

Team30: b03902024鄭筱樺 b03902084王藝霖

1 Notes

1.1 loglevel

http://elinux.org/Debugging_by_printing

The log level is used by the kernel to determine the importance of a message and to decide whether it should be presented to the user immediately, by printing it to the current console.

For this the kernel compares the log level of the message to the console_loglevel (a kernel variable) and if the priority is higher (i.e. a lower value) than the message will be printed to the current console.

```
printk(KERN_CRIT "%s, %X\n", current->comm, vmf->virtual_address);
```

Name	String	Meaning
KERN_CRIT	"2"	A critical condition occurred like a serious hardware/software failure

❏ Add loglevel to grub

```
$ sudo vim /etc/default/grub
$ GRUB_CMDLINE_LINUX_DEFAULT="quiet splash loglevel=2"
$ sudo update-grub
$ sudo reboot
```

1. 2 Recompile the kernel

❏ Recompile the kernel (but don't need to recompile the modules)

```
$ sudo make -j4 bzImage
```

❏ Install the kernel compiled by "make bzImage"

```
$ sudo make -j4 install
```

❏ Reboot ubuntu

```
$ sudo reboot
```

1.3 Test the performance (go into the compiled kernel)

❏ Clear page cache

```
echo 3 | sudo tee /proc/sys/vm/drop_caches
```

❏ Run the test code and also see how much time it spends

```
time sudo ./test
```

❏ See the syslog

```
dmesg
```

2 Trace code (15%)

Version of filemap.c: 4.6 http://lxr.free-electrons.com/source/mm/filemap.c

2.1 從mmap()這個system call開始，到filemap_fault()被設定為page fault handler的過程

// mmap: map files or devices into memory

```
int generic_file_mmap(struct file * file, struct vm_area_struct *
vma);
```

// File-backed memory regions use a generic memory region operation

```
vma->vm_ops = &generic_file_vm_ops;
const struct vm_operations_struct generic_file_vm_ops = {
```

// filemap_fault() is set to be the page fault handler

```
      .fault          = filemap_fault,
      .map_pages      = filemap_map_pages,
      .page_mkwrite   = filemap_page_mkwrite,
};
```

// Read in file data for page fault handling
// Be invoked by the vma operations vector for a mapped memory region to read in file data during a page fault.

```
int filemap_fault(struct vm_area_struct *vma, struct vm_fault
*vmf);
```

2.2 page fault發生時，是如何呼叫到filemap_fault()，以及readahead如何被執行。

❑ When page fault occurs, the system will call **`__do_fault`** (in mm/memory.c). This function call call the page fault handler **`vma->vm_ops->fault`**

```
static int __do_fault(struct vm_area_struct *vma, ...
ret = vma->vm_ops->fault(vma, &vmf);
```

❑ The page handler is **`filemap_fault`**. Readahead is performed by the following 2 functions.

```
do_async_mmap_readahead()
do_sync_mmap_readahead()
```

3 Implement pure demand paging

3.1 Notes

❏ Pure demand paging

http://www.studytonight.com/operating-system/virtual-memory

No pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults (an attempt is made to access it and that page is not already in memory).

❏ Pre-paging

Pages other than the one demanded by a page fault are brought in, i.e. readahead occurs.

❏ Load the requested page into memory

`page_cache_read()`

❏ Read ahead synchronizely: Read ahead when the required page is not in memory.

`do_sync_mmap_readahead()`

❏ Read ahead asynchronizely: Read ahead when the required page is already in memory.

`do_async_mmap_readahead()`

3.2 Implement pure demand paging(60%)

❏ comment the following 2 lines that perform readahead

```
//do_async_mmap_readahead
//do_sync_mmap_readahead
```

3.3 Result

❏ pre-paging (Default code)

```
# of major pagefault: 4201
# of minor pagefault: 2596
# of resident set size: 26680 KB

real    0m1.817s
user    0m0.020s
sys     0m0.328s
chip@chip-VirtualBox:~/hw3$
```

❏ Pure demand paging

```
# of major pagefault: 6567
# of minor pagefault: 230
# of resident set size: 26680 KB

real    0m2.874s
user    0m0.028s
sys     0m0.544s
chip@chip-VirtualBox:~/hw3$
```

3.4 Memory efficiency and runtime performance anaylsis(25%)

Memory efficiency

❏ Pre-paging: generates less pagefault, since it read ahead

❏ Pure demand paging: generates more pagefault

Runtime performance

❏ Pre-paging: spends less system time, since it generates less pagefault (access memory less)

❏ Pure demand paging: spends more system time

Part4. Bonus

❏ Adjust readahead size

vma->vm_file->f_ra.size = 1024MB/4KB

Our virtual box memory: 1024MB

Page size = 4KB

❏ We thinnk the larger readahead size will be more efficient

❏ Trace code

```
int filemap_fault(struct vm_area_struct *vma, struct vm_fault *vmf)
 struct file *file = vma->vm_file;
 struct address_space *mapping = file->f_mapping;
 struct file_ra_state *ra = &file->f_ra;

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


ra_submit(ra, mapping, file);
static inline unsigned long ra_submit(struct file_ra_state *ra,
                  struct address_space *mapping, struct file *filp)

__do_page_cache_readahead(mapping, filp, ra->start, ra->size,
ra->async_size)

int __do_page_cache_readahead(struct address_space *mapping, struct
file *filp, pgoff_t offset, unsigned long nr_to_read, unsigned long
lookahead_size)

page = page_cache_alloc_readahead(mapping);

static inline struct page *page_cache_alloc_readahead(struct
address_space *x)
__page_cache_alloc(mapping_gfp_mask(x) | __GFP_COLD | __GFP_NORETRY |
__GFP_NOWARN);
```

❏ The result is a little better. #of RSS reduces (from 26680KB to 26676KB)