Assignment 3 Design Doc: Page Replacement

Erik Andersen (Team Captain), Michael Cardoza, Yuzhuang Chen, Seongwoo Choi

Project Overview:

For this assignment, our goal is to modify the current paging implementation of FreeBSD. The purpose of this modification is to compare the current paging algorithm to our modified version of FreeBSD and analyze the difference we can see.

Project Objective:

To be more specific, our main goal is to modify vm_pageout.c where the function vm_pageout_scan() is located. In addition, we are going to modify vm_page.c and vm_phys.c and any others if need be. Note that vm_pageout_scan() is where all the pageout daemon is initiated and it seems vm_page.c and vm_phys.c is what's used to handle some queue operations. We will declare and initialize variables for logging purposes. Also, we are going to implement those variables in the modified files, which we will specify later on this document.

Page Replacement Overview:

We are going to implement a Fat Chance algorithm (found after researching fat chance) where inactive and invalid pages are placed on the front of the free list. We should check the page number, if it is even, put them on the front. If it is odd then put them on the rear. Instead of subtracting from the activity count, we will divide it by two and move it to the front of the activity list instead of the rear. Finally, when you move a page to the inactive list, it should go on the front instead of the rear. We are also going to generate some statistics about performance and write them to the system log so that we can analyze and compare between the Fat Chance Algorithm and the default paging mechanism on FreeBSD.

Files Modified:

vm_pageout.c
vm_page.h
vm_page.c
vm_phys.c
vm_phys.h

Note:

Every analysis of stats will be done in a function called vm_pageout_scan(). You can find this function under **sys/vm/vm_pageout.c** directory. In each iteration of page daemon, all the pages in active status and inactive status will be scanned. Initially, the time interval of page daemon will be set to 600 seconds, which is equivalent to 10 minutes. However, in order to run faster, we change it to 10. What we need to consider for this assignment is that we need to have counters defined so that we can keep track of stats as we are going through this assignment:

We need to consider the following throughout the assignment:

1. Number of pages scanned
2. Number of pages moved from active to inactive

3. How many pages moved from inactive to cache
4. How many pages moved from inactive to free

Modified Files:

sys/vm/vm_pageout.c

vm_pageout_scan()

In order to approach this project, we need to declare some variables inside the function vm_pageout_scan() so that we can keep track of the movement of the pages. To be specific, we need those variables to keep track of logging and statistical analysis purposes. This function, vm_pageout_scan(), is where the 'dirty' work for the page_daemon. We need to declare the following: number of pages scanned, number of pages moved from active to inactive, number of pages moved from inactive to cache, number of pages moved from inactive to free, number of pages queued for flush.

The number of pages that are scanned is incremented for each iteration of the paging system during its inactive queue. There should be loop through inactive queue and then then increment number of pages scanned by one and then end the loop procedure.

Similarly, the number of pages moved from inactive to free is incremented within the same inactive queue loop. The reason why we did this in this way is that we wanted to have the loop where the inactive queue is scanned for pages that can be moved to cache or free. Specifically, the incrementation will occur within the conditional statement.

We might duplicate some of methods we used for this assignment. This is due to the functionality of each function. For instance, a function does a certain thing, but what we want was the duplicated function does exactly the same, but the opposite of the original function.

We need to consider this function as a scanner that scans queues. Whether those queues are inactive, active, or free.

Note:

vm_pageout_scan, has been given four new variables: page_number, offset, physicalAdd, and oddoreven. The first three variables are used to find the page number of the current page. physicalAdd will use the function VM_PAGE_TO_PHYS to give us the physical address of the current vm_page. We then use a for loop that uses the page_size as the maximum length to find 2^page_size and inserts the results into offset. We then shift right the offset and divide by the physical address we found to give us the page number. Using the page_number, we do modulo 2 to help us decide if the page_number is odd or even and insert the results into oddoreven. Oddoreven is a global variable that is meant to be used by vm_phys.c to be used in the function vm_freelist_add. We also changed the when the activity count is subtracted, we changed it to simply divide by 2.

We also tracked five new variables in the function vm_pageout_scan, logged in vm_pageout_worker. See the vm_page.c section for details.

vm_pageout_init()

We need to change the time interval for page scan from the default value (for example 600 seconds, which is equivalent to 10 minutes.) to 10 seconds per specification on the assignment. In order to do this, we need to locate the default mechanism and replace the default value of 600 with 10. So, to write this in a sentence, if page update period is zero, then set the time interval to 10 seconds and then end the if statement. We also made a decision to add new variables to the code so that we can use it to determine inactive, active, and free queues.

Pseudocode:

vm_pageout.c **line 944**

The variables are made to be used to find page number and to have a global variable of oddoreven.

```
int oddoreven;
vm_pageout_scan(){
        Long page_number;
        Long offset;
        vm_paddr_t physicalAdd;
        …
```

vm_pageout.c **line 1031**

This line of code makes the page number and checks to see if the page number is odd or even.

```
        ...
        for(){ //inactive
                physicalAdd = page's_physical address;
                offset = 2^PAGE_SIZE;
                physicalAdd = physicalAdd >> offset;
                physicalAdd = physicalAdd / PAGE_SIZE;
                oddoreven = page_number%2;
                …
```

vm_pageout.c **line 1846**

This line of code divided act_count by 2.

```
...
m->act_count = m->act_count/2;
...
```

vm_pageout.c **line 1862**

This line of code makes the vm_pageout_update_period 10 instead of 600

```
vm_pageout_init(){
        ...
        if (vm_pageout_update_period == 0)
```

```
            vm_pageout_update_period = 10;
        …
}
```

vm_pageout.h **line 79**
Makes int oddoreven a global variable to be used by freelist_add.

```
…
extern int oddoreven;
…
```

vm_pageout.h **line 79**
Makes int oddoreven a global variable to be used by freelist_add.

```
…
extern int oddoreven;
...
```

sys/vm/vm_phys.c

We need to place inactive and invalid pages on the front of the free list if the page number is even and to the rear if the page number is odd.
On vm_freelist_add, we have a simple for loop to check if odd or eveisodd or even. If it's even we insert the page to the front and if it's odd, we insert it to the tail.

Pseudocode:
vm_phys.c **line 248**
This line of code will send a page to the back of the queue if it is a odd or to the front if its even.

```
vm_freelist_add(){
        ...
        if(odd){
                Page to tail;
        }
        if(even){
                Page to head;
        }
        ...
}
```

sys/vm/vm_page.c

vm_page_enqueue, is in charge of inserting a specified page into a either the rear of the active queue. To change this into for the requirements of "Fat Chance", we just changed the tailq from inserting to the tail to the head.

_vm_page_deactivate, was changed to ensure that the new element was thrown into the head instead of the tail.

Five new variables were defined to track the values we wanted to log. These are declared as extern in vm_page.h and initialized to 0 in vm_page.c. They are as follows: scan_count, inactive_moved, inactive_cache, inactive_free, flush_num. These values are tracked in vm_pageout.c as follows.

Scan_count: Each time a page is scanned, this value is incremented
Inactive_moved: Moving a page from the active to inactive queue increments this number
Inactive_cache: Moving a page from the inactive to cache queue increments this number
Inactive_free: Moving a page from the inactive to free queue increments this number
Flush_num: This number is incremented when a page is queued for flush

Pseudocode:

vm_page.c **line 2442**
This line of code makes sure that the page will go to the head of the inactive queue.

```
_vm_page_deactivate(){
        ...
        if(athead){
                TAILQ_INSERT_HEAD();
        }else{
                TAILQ_INSERT_HEAD();
        }
        ...
}
```

vm_page.c **line 2112**
This line of code makes sure the page goes to the front of the active queue.

```
vm_page_enqeue(){
        …
        TAILQ_INSERT_HEAD();
        …
}
```

Testing

Writeup.pdf will be provided for a detailed explanation of the test procedures, hypothesis and finding and some more detailed analysis of algorithm we used for creating the testing.

Now, this is the end of design doc. You are now allowed to switch the document to the write up.