This is the design document for Enes Yazgan and Ian Feekes.

Table of Contents:

1. Things To Note

This project had only 2 cases.

Case 1:

The default Page Replacement Algorithm.

Case 2:

Our FIFO Page Replacement Algorithm.

NOTE: Both files were also modified (in the same way) to log the following into /var/log/messages:

- Each page scanned
- Each page moved from the active to inactive queue
- Each page moved from the inactive queue to cache/ or freed
- Each page queued for flush

This change was only to aid us in benchmarking, it has no performance changes.

To test out modified code, simply:

1. sudo cp ../Case\ 1/vm_pageout.c /usr/src/sys/vm/
2. sudo make -DKERNFAST buildkernel
3. sudo make installkernel

2. What We Did

First, in order to be able to log our messages (explained in previous section), we included the following into vm_pageout.c

```
#include <sys/syslog.h>
```

We then defined the following variables in the function, vm_pageout_scan()

1. Pages_scanned //number of pages scanned in the function
2. atoi //number of pages moved from active to inactive
3. itoc //number of pages moved from inactive to cache
4. itof //number of pages moved from inactive to free
5. qforf //number of pages queued for flush

These variables were incremented in the following ways:

In the Inactive loop of vm_pageout_scan():

Pages_scanned is incremented by 1 for each page scanned in the loop

itof is incremented by one for each invalid page freed within this loop

itoc is incremented by 1 in the free_page: section at the end of the loop, for when a page is clean.

qforf is incremented by 1 otherwise.

In the Active loop of vm_pageout_scan():

Pages_scanned is incremented by the number of pages that have been scanned by the end of the loop.

atoi is incremented by 1 when a page is deactivated on the event of

a. We are not short for inactive pages
b. Current page is clean

At the end of these two loops, we simply log these variables values into /var/log/messages so that we can use the values for our benchmarks later on.

In order to implement FIFO Page Replacement, we modified vm_pageout_scan() again. We realized that the inactive and active loops in this scan were modifying the positions of the pages following the CLOCK/Activity method, and decided to simply change this code to operate upon the logic of FIFO, effectively forcing these two queues to act as FIFO queues.

What we changed in the function vm_pageout_scan() is the following:

In the Inactive loop:

Remove all sections of code that modify act_delta, reference the ref_count attribute, or the act_count attribute.

In the section of the Inactive loop that decides whether or not to activate a page, instead of doing so for all elements of the queue with act_delta of !0, we instead only did so for pages at currently at the head of the queue, with if(m==TAILQ_FIRST(&pq->pq_pl)){}

This was done to ensure that things only left the inactive queue from the head, preserving the FIFO paradigm.

We acted similarly in the Active queue, allowing pages to deactivate if and only if they were currently at the head of the active queue.

Upon inspection of vm_page.c and vm_page.h, we realized that the functions adding the pages to the inactive and active queues were adding them to the tail of the queues, and thus did not need any modifications of their own. The files vm_page.c and vm_page.h remain unchanged.

3. How We Tested It

Using the following command:

grep memory /var/run/dmesg.boot

Showed us that our VM's had the following memory available:

Real (4608M)
Available(3907M)

Despite trying the given memorytest.c, the max cap of 2000M for the test left our benchmarks lackluster, as it didn't push our memory to the limit.

We insteal looked online and found a test called stress.
To install it, we used:

pkg install stress

To run it:

stress -m XX --vm-bytes YYM -t ZZs

-m XX = XX hogs repeating malloc()/free()

--vm-bytes YYM = YYM per hogs
-t ZZs = timeout in ZZs

With this test, we were able to really push our system. These are the specific tests we ended up using:

Test1: stress -m 1 --vm-bytes 4000M -t 20s
Test2: stress -m 5 --vm-bytes 815M -t 20s

In order to view our results, and convert them into a legible set of data, we got our logged values from /var/log/message, and used these values to generate graphs, which are viewable and explained in WRITEUP.pdf

4. What We Got

Refer to the included WRITEUP.pdf for our findings