**Please note that you will find the changes in duplicated methods highlighted**

**Vm\_phys.h:**

Adding parameter in vm\_physic\_free\_ir() and set its default to 0 :

1. **void** vm\_physic\_free\_ir (vm\_page\_t m)

**vm\_phys.c**

1. vm\_phys\_free\_pages\_ir(vm\_page\_t m, **int** order, int flag=0)
2. {
4. fl = (\*seg->free\_queues)[m->pool];
5. **if**(flag==0){
6. vm\_freelist\_add(fl, m, order, 1);
7. }
8. **if**(flag==1)
9. vm\_freelist\_add(fl, m, order, 0);
10. }

**vm\_page.h**

1. Adding pages counters to keep tracking pages’ movements
2. **struct** vm\_domain {
3. **struct** vm\_pagequeue vmd\_pagequeues[PQ\_COUNT];
4. u\_int vmd\_page\_count;
5. u\_int vmd\_free\_count;
6. u\_int scanned\_active\_pages\_counter;
7. u\_int scanned\_inactive\_pages\_counter;
8. u\_int cached\_pages\_counter;
9. u\_int flushed\_pages\_counter;
10. u\_int reactivated\_pages\_counter;
11. u\_int deactivated\_pages\_counter;
13. **long** vmd\_segs;  /\* bitmask of the segments \*/
14. boolean\_t vmd\_oom;
15. **int** vmd\_pass;   /\* local pagedaemon pass \*/
16. **struct** vm\_page vmd\_marker; /\* marker for pagedaemon private use \*/
17. };

1. Adding new function prototype to add deactivate page in the front instead of the rear .
2. vm\_page\_deactivate\_if (vm\_page\_t m)

3. Adding vm\_page\_free\_r(vm\_page\_t m) prototype

1. **void** vm\_page\_free\_ir(vm\_page\_t m)
2. Adding vm\_page\_free\_toq\_r(vm\_page\_t m )
3. vm\_page\_free\_toq\_ir(vm\_page\_t m )
4. Adding vm\_page\_requeue\_locked\_if(vm\_page\_t m ) to move the page to the front of current page instead of the tail
5. **void** vm\_page\_requeue\_locked\_if(vscannm\_page\_t m)

**vm\_page.c:**

1. **Declare vm\_page\_free\_r(vm\_page\_t m) method ; within it call vm\_page\_free\_toq\_r() :**
2. vm\_page\_free\_r(vm\_page\_t m) {
3. m->flags &= ~PG\_ZERO;
4. vm\_page\_free\_toq\_r(m);
5. }
6. **Declare vm\_page\_deactive\_r which would insert page in front of the queue by calling \_vm\_page\_deactivate and switch its parameter from 0 instead of 1**
7. vm\_page\_deactivate\_r(vm\_page\_t m)
8. {
10. \_vm\_page\_deactivate(m, 1);
11. }

**3. Declare vm\_ \_free\_toq\_ir insert page into the rear instead of the front by calling**

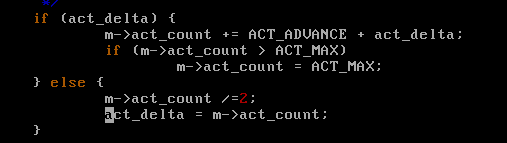
1. **void**
2. vm\_page\_free\_toq\_ir(vm\_page\_t m)
3. {
5. **if** ((m->oflags & VPO\_UNMANAGED) == 0) {
6. vm\_page\_lock\_assert(m, MA\_OWNED);
7. KASSERT(!pmap\_page\_is\_mapped(m),
8. ("vm\_page\_free\_toq: freeing mapped page %p", m));
9. } **else**
10. KASSERT(m->queue == PQ\_NONE,
11. ("vm\_page\_free\_toq: unmanaged page %p is queued", m));
12. PCPU\_INC(cnt.v\_tfree);
14. **if** (VM\_PAGE\_IS\_FREE(m))
15. panic("vm\_page\_free: freeing free page %p", m);
16. **else** **if** (vm\_page\_sbusied(m))
17. panic("vm\_page\_free: freeing busy page %p", m);
19. /\*
20. \* Unqueue, then remove page.  Note that we cannot destroy
21. \* the page here because we do not want to call the pager's
22. \* callback routine until after we've put the page on the
23. \* appropriate free queue.
24. \*/
25. vm\_page\_remque(m);
26. vm\_page\_remove(m);
28. /\*
29. \* If fictitious remove object association and
30. \* return, otherwise delay object association removal.
31. \*/
32. **if** ((m->flags & PG\_FICTITIOUS) != 0) {
33. **return**;
34. }
36. m->valid = 0;
37. vm\_page\_undirty(m);
39. **if** (m->wire\_count != 0)
40. panic("vm\_page\_free: freeing wired page %p", m);
41. **if** (m->hold\_count != 0) {
42. m->flags &= ~PG\_ZERO;
43. KASSERT((m->flags & PG\_UNHOLDFREE) == 0,
44. ("vm\_page\_free: freeing PG\_UNHOLDFREE page %p", m));
45. m->flags |= PG\_UNHOLDFREE;
46. } **else** {
47. /\*
48. \* Restore the default memory attribute to the page.
49. \*/
50. **if** (pmap\_page\_get\_memattr(m) != VM\_MEMATTR\_DEFAULT)
51. pmap\_page\_set\_memattr(m, VM\_MEMATTR\_DEFAULT);
53. /\*
54. \* Insert the page into the physical memory allocator's
55. \* cache/free page queues.
56. \*/
57. mtx\_lock(&vm\_page\_queue\_free\_mtx);
58. m->flags |= PG\_FREE;
59. vm\_phys\_freecnt\_adj(m, 1);
60. #if VM\_NRESERVLEVEL > 0
61. **if** (!vm\_reserv\_free\_page(m))
62. #else
63. **if** (TRUE)
64. #endif
65. vm\_phys\_free\_pages\_r(m, 0);
66. **if** ((m->flags & PG\_ZERO) != 0)
67. ++vm\_page\_zero\_count;
68. **else**
69. vm\_page\_zero\_idle\_wakeup();
70. vm\_page\_free\_wakeup();
71. mtx\_unlock(&vm\_page\_queue\_free\_mtx);
72. }
73. }
74. **Declear vm\_page\_requeue\_locked\_if to insert the page in the front of its current qeueu instead of the rear**
75. **void**
76. vm\_page\_requeue\_locked\_if(vm\_page\_t m)
77. {
78. **struct** vm\_pagequeue \*pq;
80. KASSERT(m->queue != PQ\_NONE,
81. ("vm\_page\_requeue\_locked: page %p is not queued", m));
82. pq = vm\_page\_pagequeue(m);
83. vm\_pagequeue\_assert\_locked(pq);
84. TAILQ\_REMOVE(&pq->pq\_pl, m, plinks.q);
85. TAILQ\_INSERT\_HEAD(&pq->pq\_pl, m, plinks.q);
86. }

Init counter with zero

1. **static** **void**
2. vm\_page\_domain\_init(**struct** vm\_domain \*vmd)
3. {
4. **struct** vm\_pagequeue \*pq;
5. **int** i;
7. \*\_\_DECONST(**char** \*\*, &vmd->vmd\_pagequeues[PQ\_INACTIVE].pq\_name) =
8. "vm inactive pagequeue";
9. \*\_\_DECONST(**int** \*\*, &vmd->vmd\_pagequeues[PQ\_INACTIVE].pq\_vcnt) =
10. &cnt.v\_inactive\_count;
11. \*\_\_DECONST(**char** \*\*, &vmd->vmd\_pagequeues[PQ\_ACTIVE].pq\_name) =
12. "vm active pagequeue";
13. \*\_\_DECONST(**int** \*\*, &vmd->vmd\_pagequeues[PQ\_ACTIVE].pq\_vcnt) =
14. &cnt.v\_active\_count;
15. vmd->vmd\_page\_count = 0;
16. vmd->vmd\_free\_count = 0;
17. vmd->scanned\_active\_pages\_counter = 0;
18. vmd->scanned\_inactive\_pages\_counter = 0;
19. vmd->deactivated\_pages\_counter = 0;
20. vmd->reactivated\_pages\_counter = 0;
21. vmd-> cached\_pages\_counter = 0;
22. vmd->vmd\_flushed = 0;
23. vmd->vmd\_segs = 0;
24. vmd->vmd\_oom = FALSE;
25. vmd->vmd\_pass = 0;
26. **for** (i = 0; i < PQ\_COUNT; i++) {
27. pq = &vmd->vmd\_pagequeues[i];
28. TAILQ\_INIT(&pq->pq\_pl);
29. mtx\_init(&pq->pq\_mutex, pq->pq\_name, "vm pagequeue",
30. MTX\_DEF | MTX\_DUPOK);
31. }
32. }

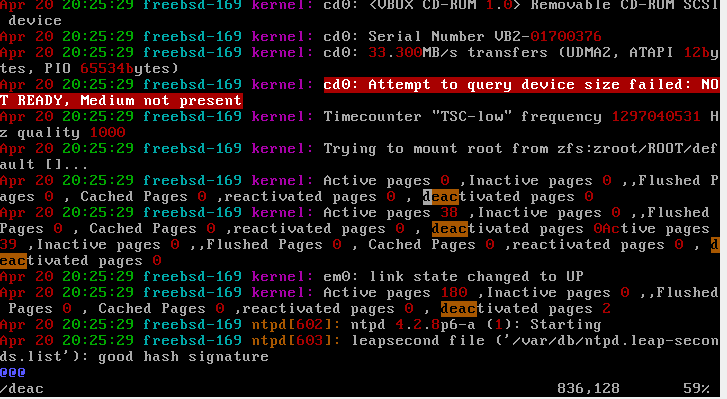
vm\_pageout.c

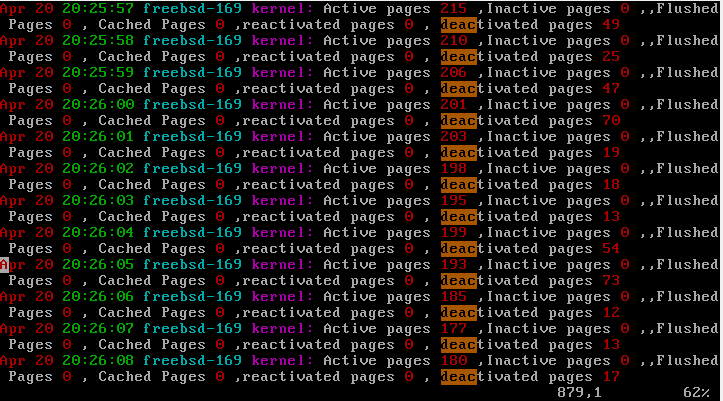
1. Change update\_period to 10 seconds instead of 600 second
2. **if** (vm\_pageout\_update\_period == 0)
3. vm\_pageout\_update\_period = 10;
4. **In order to divided the activity count by 2 instead of subtracting**

****

1. **Create static function that print how many pages went from active to and from inactive to caches ,besides the flashed pages .**

****

****

****