Project2_Milestone3 Memory Management Report

Yukui Ye
SUID: 439644268

1. Comment on the new implemented policy(the pros and cons)

Cons: If some page is heavily used, its NFU score will be bigger, but if the page is not used ever again or rarely used, the page will never be chosen for replacement due to high score since system only swap out pages with lower score than threshold. Therefore, old process running for a while reaches its maximum NFU score, then stop executing, which turns to be useless page, will stay in memory forever, and if new process first comes in the system with 0 NFU score (less than initial threshold), it will immediately be swapped out. This situation will lead to waste amount of memory space store pages that rarely used. Probably it will reaches an extremely situation, which will crash the whole memory system at the very end, due to old useless process stay in the memory and new process can not come in.

Pros: It favors the program that running long time, the best is never stop, which can not happen in the real world.

In sum, NFU algorithm is theoretical not practical.

2. Experiment for comparing the original policy and new policy

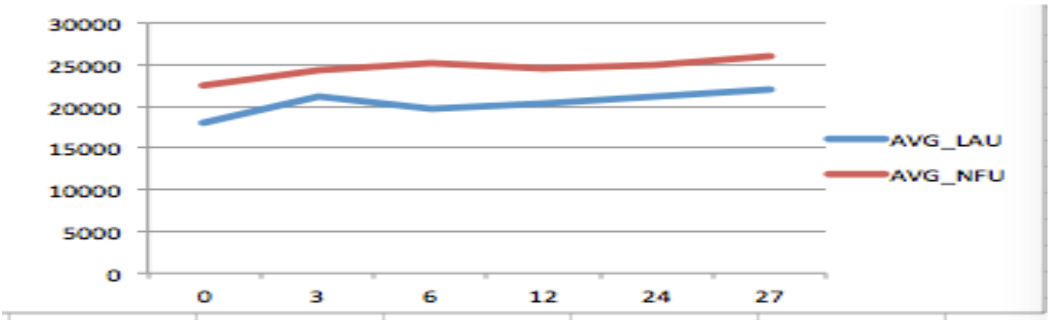
Configuration: MEMRUN:Adjusting Large/Small Working Set, Adjust VMware physical memory to 52M. For MEMRUN we set parameter to (0 2000) which means for 2000 seconds, we only request pages from a small working set. When enable NFU, initial threshold is 1.

```
./memrun  0 2000
```

Table : Data recording for Task1

Minute	LAU				NFU			
	1	2	3	AVG_LAU	1	2	3	AVG_NFU
0	17988	17999	18004	17997	22510	22512	22512	22511
3	19460	21847	22070	21126	24481	24465	24378	24441
6	19844	19605	19537	19662	25316	25315	25136	25256
12	19656	21684	19503	20281	25986	23879	23688	24518
24	20452	23196	19729	21126	24533	24732	25506	24924
27	22587	22921	20524	22011	25470	25517	26881	25956

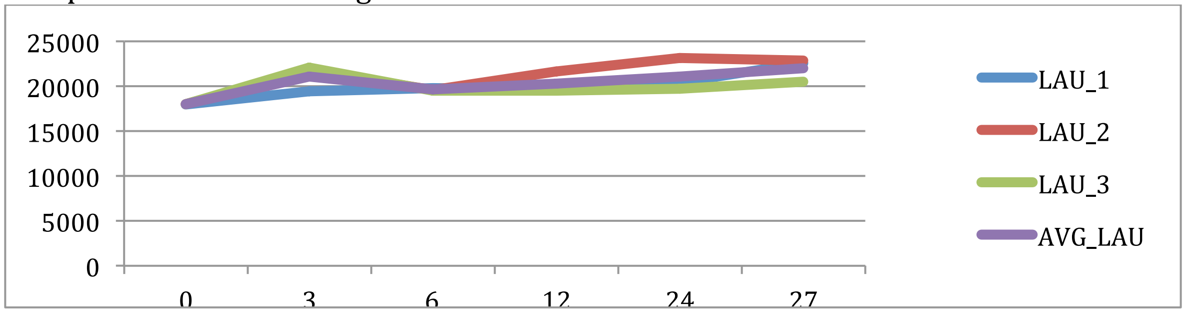
Graph based on the data :

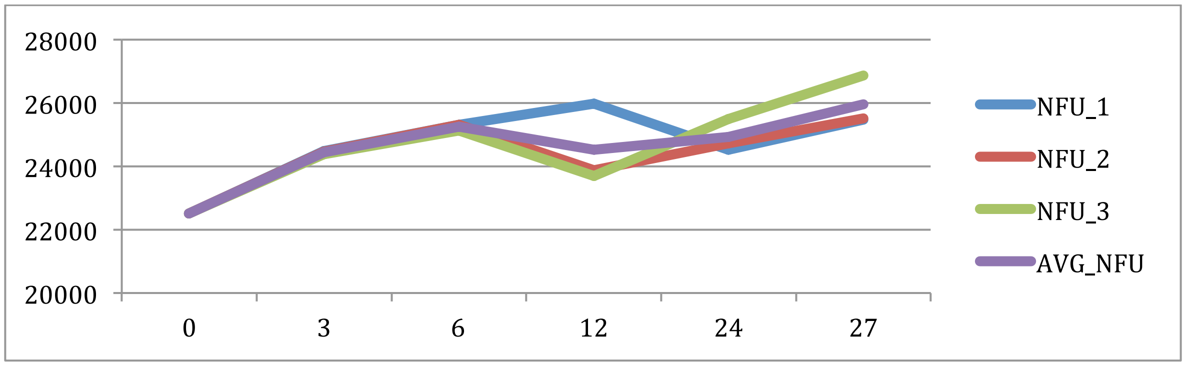


Experiment analysis:

Comparison between different algorithm: we can tell from the graph, AVG_NFU line is higher than AVG_LAU line, which means the average of the number of the total page faults of NFU is bigger than that of the LAU algorithm. That is because when enable the NFU algorithm, the memory test program comes into the system with 0 threshold, while the initial threshold is 1, so it bellows the threshold, a lot of pages of the test program would be kicked out, therefore the total number of page faults increased bigger than original algorithm. Besides from the graph, the total page faults number of NFU has a relatively smoothly increasing as time goes longer.

Comparison within one algorithm at different minutes:





Experiment analysis:

When requesting a small number of pages during 2000 seconds, the changes within different minutes for LAU: no matter how long it waits, 3 or 6 minutes, it has no big change. While for NFU: within longer waiting time, the total number of page faults will increased or dropped relatively bigger than LAU. I think in another words, NFU change faster than LAU. That is because when enable NFU, the program comes into system with 0 threshold which is less than the initial threshold 1, so a lot of pages being kicked out, therefore lead to a large number increased in the total number of page faults, then after stop the program, the number has a relatively big dropped, when enable it again, it has relatively dramatically increased, after execute the program back and forth, the total number of page faults has relatively big changed.

Interesting Observation: from the first graph, the AVG_LAU vs AVG_NFU, the blue line (AVG_LAU) has a relatively big increased at 3th minute, which is when we start to run the memrun test, then it increased relatively smoothly at later time. I think that is because when one big change coming to the system at the first time, maybe the system is not used to it, so it changed bigger than the run it the second or third time.

3. Experiment on the new policy

Configurations: MEMRUN:Adjusting Large/Small Working Set, Adjust VMware physical memory to 52M. Enable NFU algorithm, and follow the instruction to set initial threshold at 1,8,15,20,21(following screenshot is the last test for task2 with initial threshold 21)
gcc –o proc1 memrun.c; gcc –o proc2 memrun.c
./Proc1 2000 0 (for 2000 seconds make a lot of random page requests to access a large number of pages)
./Proc2 20 2000 (for 20 seconds request a large number of pages, then for later 2000 seconds request a small number of pages.)

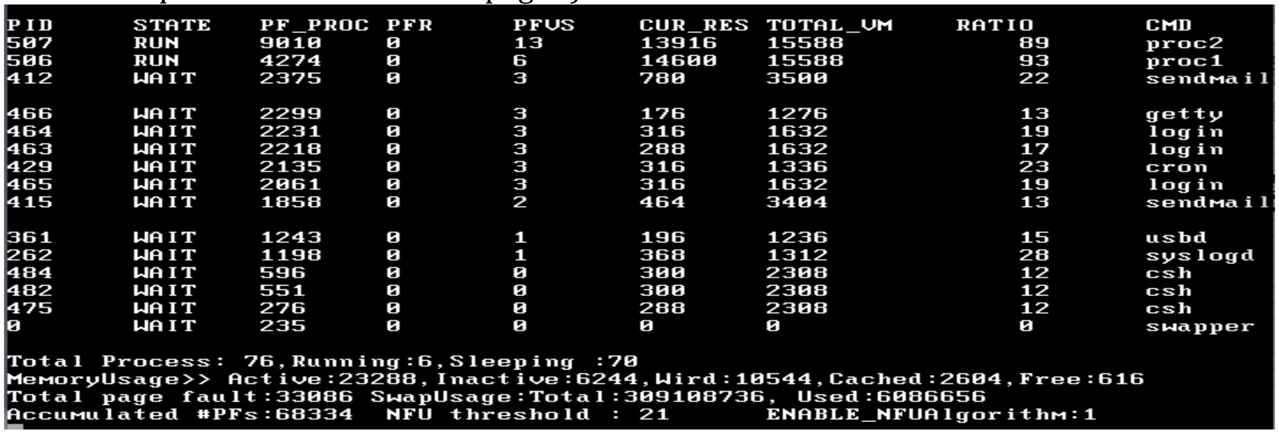
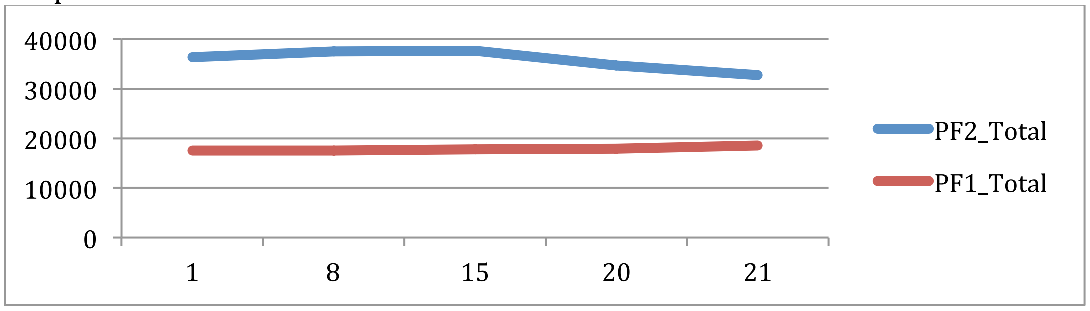

```
PID     STATE    PF_PROC  PFR     PFVS     CUR_RES  TOTAL_VM     RATIO       CMD
507     RUN      9010     0       13       13916    15588           89       proc2
506     RUN      4274     0       6        14600    15588           93       proc1
412     WAIT     2375     0       3        780      3500            22       sendmail

466     WAIT     2299     0       3        176      1276            13       getty
464     WAIT     2231     0       3        316      1632            19       login
463     WAIT     2218     0       3        288      1632            17       login
429     WAIT     2135     0       3        316      1336            23       cron
465     WAIT     2061     0       3        316      1632            19       login
415     WAIT     1858     0       2        464      3404            13       sendmail

361     WAIT     1243     0       1        196      1236            15       usbd
262     WAIT     1198     0       1        368      1312            28       syslogd
484     WAIT     596      0       0        300      2308            12       csh
482     WAIT     551      0       0        300      2308            12       csh
475     WAIT     276      0       0        288      2308            12       csh
0       WAIT     235      0       0        0        0                0       swapper

Total Process: 76,Running:6,Sleeping :70
MemoryUsage>> Active:23288,Inactive:6244,Wird:10544,Cached:2604,Free:616
Total page fault:33086 SwapUsage:Total:309108736, Used:6086656
Accumulated #PFs:68334  NFU threshold : 21       ENABLE_NFUAlgorithm:1
```

Table : Data recording for Task2

Threshold	NFU												
	Proc1				Proc2				PF2_Total				PF1_T
	1	2	3	AVG	1	2	3	AVG	1	2	3	AVG	
1	6414	6239	10088	7580	8790	11090	8864	9581	34047	36544	38506	36366	17592
8	5744	4304	10036	6695	11408	7560	15931	11633	35812	31079	45667	37519	17558
15	6112	4016	6672	5600	11046	9076	17590	12571	36186	32567	44200	37651	17861
20	3906	4744	3877	4176	9084	7883	8817	8595	34318	34587	35110	34672	17921
21	3990	4420	4274	4228	10761	7932	9010	9234	33611	31664	33086	32787	18631

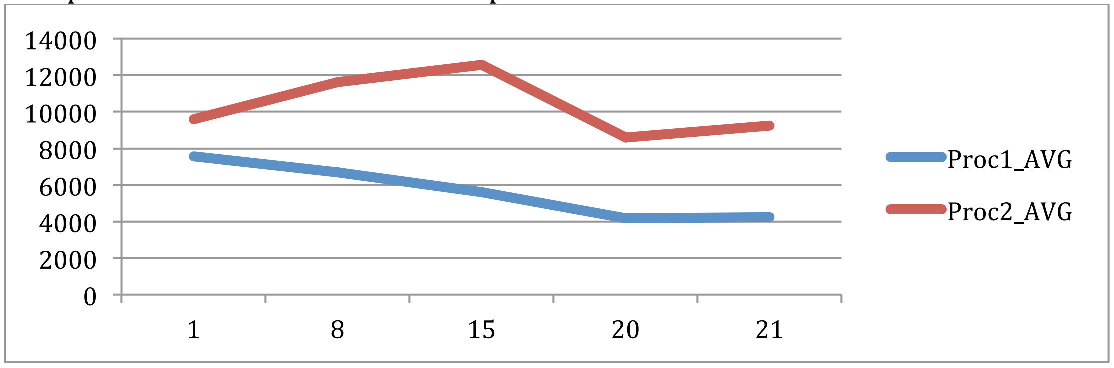
Graph based on data:



From the graph, we can see that PF2_Total is much higher than PF1_Total, which means after we run the two processes, the total number of page faults increased dramatically, that is because when start to run the two processes, the two comes in the system with threshold 0, which is less than initial threshold, they will occupy the memory and request large number of pages, which will definitely lead to large number of page faults happened.

Interesting observation: In the graph, the blue line which represent PF2_Total seems reaches the highest point with the initial threshold 15, after that it drop down, and it reaches the lowest point with initial threshold 21. Besides, it seems almost have the same amount of page faults with initial threshold 20 and 21. I think this situation happened because the maximum NFU score of each process in the system is 20, when initial threshold equal to 20 or larger than 20, the system swapped out a lot of pages whose score is lower than initial threshold, after a while, old process in active list that is not currently running will be swapped out, and new useful pages come in, therefore the total number of page fault will drop down.

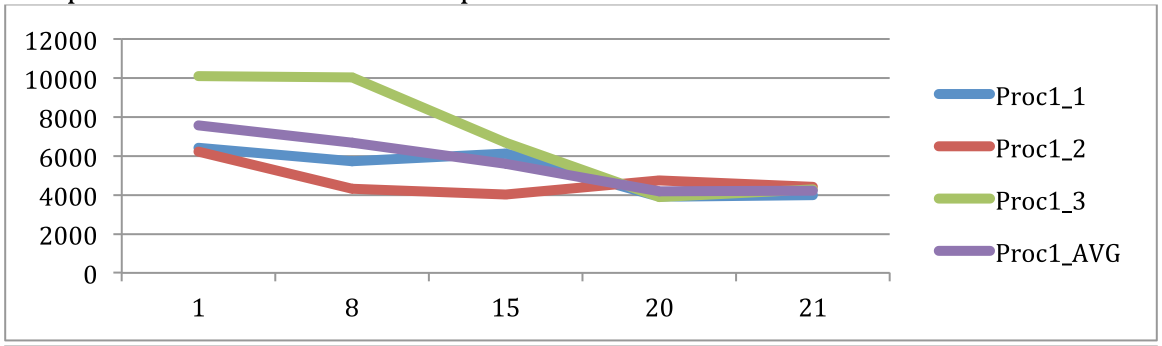
Comparison between two different processes:

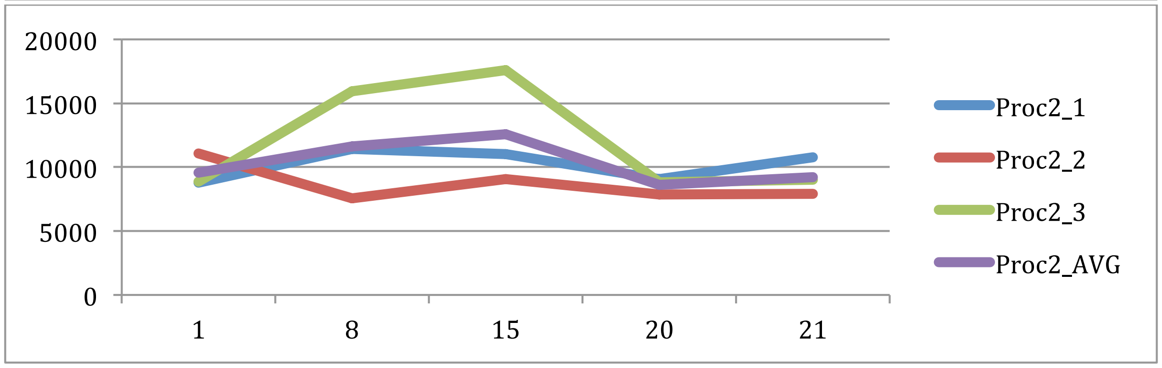


Proc1 2000 0 ; Proc2 20 2000.

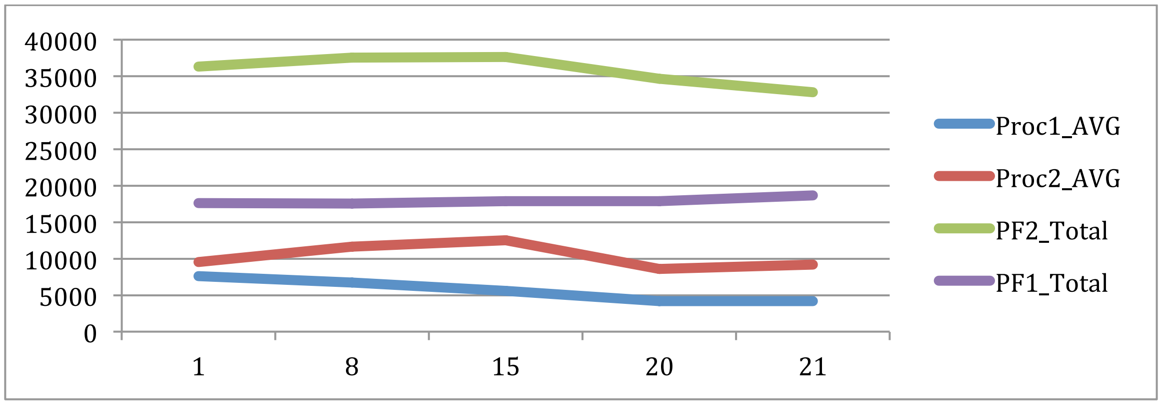
From the above graph, the total page fault number of prcess2 is higher than that of process1. At the beginning 20 seconds, the two processes all make a lot of random request to access a large number of pages, but later, Proc2 request pages from a small working set for 2000 seconds, while Proc1 keep request a large number of pages as before. Total number of page fault for Process1 has a smoothly decrease as initial threshold getting larger. Besides, from graph, the blue line at initial threshold 20 and 21,it did not change anymore. That is because when we set the initial threshold to the maximum threshold, after a running for a while, the total number of page fault will not decrease as before, it will keeps at one certain number, which reaches the lowest number during the changes.

Comparison within two different processes:





From above two graph, process2 has a big change within itself comparing to process1, I think this is because there are two different policy during the executing of prcess2, it will change to request pages from small working set after 20 seconds random requesting pages from large set. So the number will have obviously changed (increased or decreased) when it changed to another policy. For process1 which continued keep the same policy(request a large number of pages randomly), it smoothly drop down at certain number which is the point that reaches the maximum threshold.



From the graph, I find that when we plus together the total number of page fault of process1 and process2, it still less than the total page fault number of the system. There is still other process either running or waiting in the active list cause page fault.

Interesting observation: when we set the initial threshold to 1, after we run the two process, I find that the threshold number double jumped so quickly to 16 and then back to 1 then 4,8,16…

it changed so fast, while when I set the initial threshold to 8, 15,20,21, actually the threshold did not jump so frequent compare to the one with number 1 as initial threshold.

4. Comparison of policies

LAU: First compute the number of pages we want to move from active queue to the inactive queue recorded in page_shortage, then scan the active queue for things we can deactivate by tracking the per-page activity counter. If the activity counter equal to 0, then we deactivate it from active list.

NFU: Every page has its own NFU score, which increased by one whenever it has been referenced. The page will be swapped out when its NFU score less than dynamically changed threshold.

From the discussion in Part1, we already know the property of NFU, either its pros or cons. Compare to NFU, LAU is much better and more practical. It only pages out the pages that has no referenced bits, which means only page out old pages with least active used in order to make more space for the new useful pages. Make system more efficiency.