CS 1550: Project 3

Instructed by *Prof. Jonathan Misurda*

Grader: *Zhenjiang Fan*

Due on Nov. 19, 2017

Zac Yu (LDAP: zhy46@)

November 19, 2017

1 VM Simulator Report

1.1 Abstract

An integral part of the memory management logic of a modern Operating System is the page replacement algorithm, the algorithm to decide which page to evict out of the physical memory when it is full. Depending on the algorithm of choice, the performance (measured by the total number of page faults and number of dirty page writes) vary. In this report, we explore the performance of four different page replacement algorithms (namely "Opt," "Clock," "NRU," and "Random"). We do so by implementing a page table for and simulate page operations from recorded trace files of actual program execution. Finally, we speculate on the most preferable page replacement algorithm for modern OSs based on our observations.

1.2 Implementation Overview

1.2.1 Page Table

We implemented our simulated page table using a inverted page table structure (an array of simulated Page Table Entries) with cache (a hash map). We assume a 32-bit address space with each page containing $2^12 = 4096$ addresses (4 KB in size), and test with 8, 16, 32, and 64 frames respectively. Since the space complexity for our simulation is not a concern, we don't have to limit the size of our cache. Each inverted page table entry contains a page ID, a D (dirty) bit, and a R (referenced) bit. The page table exposes two public functions, read and write, to the user to allow simulating page read and write operations respectively. For both operation, it performs a page ID lookup in the frame (via the cache) and loads/swaps the page into the physical memory (represented by the inverted table) when necessary. It updates the dirty bits on write operations and maintains the metrics (see section 1.3.1) for performance measurement. Each page table is initialized with a given number of frames and an evictor (see section 1.2.2).

1.2.2 Abstract Evictor

When a page swap is necessary, some page replacement algorithm is responsible for choosing a page to evict out of the physical memory. We abstract the functioning unit of a page replacement algorithm by an evictor. As the name suggested, it provides a public function to suggest the page (represented by a frame ID) to evict at any given time during the simulation. In addition, it has a callback function to be called after each operation in the page table to maintain some property (e.g. to set the R bit).

1.2.3 Optimal Evictor (`opt`)

The optimal evictor is initialized with the complete sequence of operations and precomputes the time when a page is going to be referenced next (infinity if it is not going to be used again) for each operation. During the simulation, it maintains an array of times that corresponds to when each page (load in frame) is going

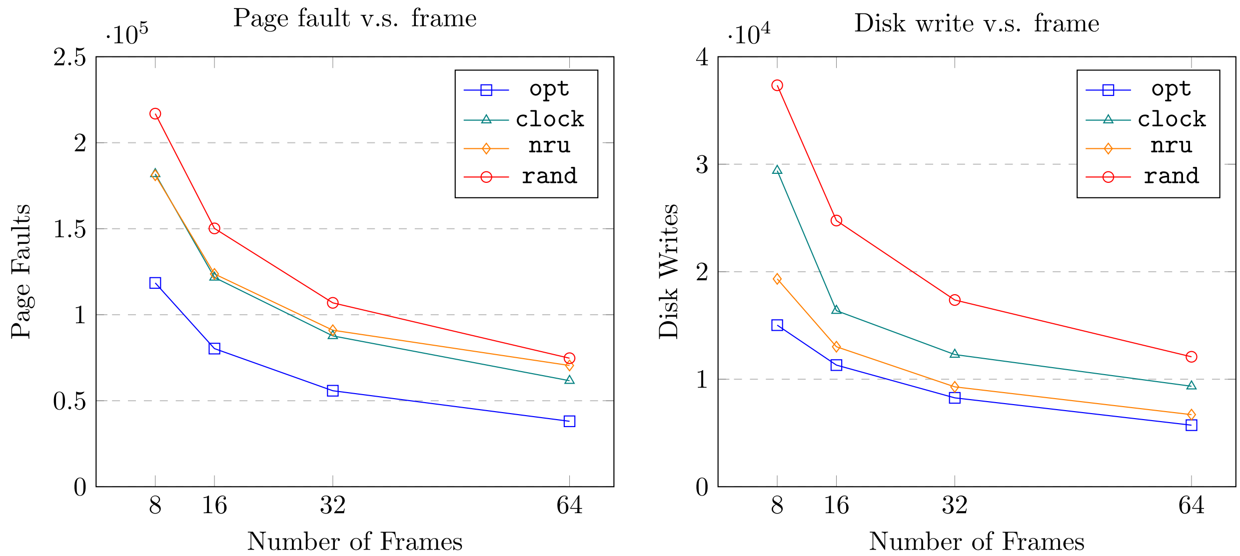

Figure 1: `gcc.trace` page faults and disk writes

to be referenced next. When it is requested to suggest an evictee, it simply returns the index that has the largest value in the referenced-next array. The R bit is not used for this algorithm.

1.2.4 Second Chance (Clock) Evictor (`clock`)

The clock evictor maintains a pointer to the frame ID to be considered as evictee next. When it is requested to suggest an evictee, it cycle through the frames, set referenced pages to be unreferenced, and return the index of the first referenced page. It also updates the R bit after each operation.

1.2.5 Not Recently Used (NRU) Evictor (`nru`)

The NRU evictor uses a counter as a timer to clear all R bits periodically, subject to the refresh rate it is initialized with. During the simulation, after each operation, it increments the counter value, clear the R bits when appropriate, and updates the R bit of the newly referenced page to 1. When it is requested to suggest an evictee, it finds the frame that is loaded with a page with the minimum $(\texttt{RD})_2$ value.

1.2.6 Random Evictor (`rand`)

The random evictor always returns a random (loaded) frame ID for evictee.

1.3 Performance Analysis

1.3.1 Metrics

We used the total number of page faults that occur during the simulation (Page Faults) and the total number of dirty frames that has to be written back to disk (Disk Writes) to measure the performance of the algorithms. Both metrics above can be implemented as variable counters in the simulated page table that increment when the corresponding actions take place.

1.3.2 Performance Across Algorithms

Table 1 lists the counter values, for each trace file, for each number of frames (n), for each algorithm (using a refresh rate of $4n$ for NRU).
As shown in Figure 1, when simulating `gcc.trace`, the optimal algorithm, as expected, yielded the fewest

Frames	Algo	Page Faults	Disk Writes
8	opt	118480	15030
	clock	181856	29401
	nru	181092	19333
	rand	216898	37343
16	opt	80307	11314
	clock	121682	16376
	nru	123710	13022
	rand	150189	24760
32	opt	55802	8266
	clock	87686	12293
	nru	90990	9289
	rand	106898	17369
64	opt	38050	5725
	clock	61640	9346
	nru	70481	6705
	rand	74716	12086

Metrics for `gcc.trace`

Frames	Algo	Page Faults	Disk Writes
8	opt	171244	46449
	clock	293519	54327
	nru	294885	51943
	rand	321308	54500
16	opt	78312	18129
	clock	191848	48350
	nru	165840	29760
	rand	194614	40082
32	opt	28826	6899
	clock	53025	11140
	nru	61518	7778
	rand	84050	18376
64	opt	14289	4097
	clock	22611	5844
	nru	39402	4707
	rand	35244	8458

Metrics for `swim.trace`

Table 1: Performance statistics across algorithms

number of page faults and disk writes across all configurations. Conversely, the random algorithm causes more page faults and disk writes for most configurations.

We observed that the number of page faults resulted in by the clock algorithm and the NRU algorithm are close. However, due to clock algorithm's lack of consideration for disk writes, it generally causes significantly more disk writes when compared to the NRU algorithm.

It's interesting to note that the when the refresh rate is set to be strictly promotional to the number of frames, the performance of the NRU algorithm degrades over number of frames. Specifically, with 32 or 64 frames in the physical memory, the NRU algorithm's number of page faults becomes noticeably larger than that of the clock algorithm. This problem can be mitigated by finely adjusting the refresh rate accordingly to the number of frames (more in section 1.3.3).

In Figure 2, we observed a similar trend for `swim.trace`. However, the performance measured by the number of page faults degrades even faster for the NRU algorithm. With 64 frames in the physical memory, the NRU algorithm performs even worse than the random algorithm (namely choosing pages to evict randomly).

In addition, we noticed that the number of disk writes of the clock algorithm is inconsistent. In particular, with 8 or 16 frames, the clock algorithm resulted in close to or even more disk writes than the random algorithm. Indeed, since the dirty bit is not used deciding the page to evict, there is no guarantee whatsoever.

1.3.3 Performance Across Refresh Rate

Table 2 lists the counter values, for each trace file, for each number of frames (n), for each refresh rate (out of 16, 32, 64, 128, 256, 384, and 512) for the NRU algorithm.

As shown in Figure 3, the number of page faults with the NRU algorithm depends on the number of frames and the refresh rate. For instance, for the `gcc.trace` file, the optimal refresh rate around 16 (2x of the number of frames) for 8 frames, around 64 (4x) for 16 frames, around 128 (4x) for 32 frames, and around 512 (8x) for 64 frames. However, for the `swim.trace` file, the optimal refresh rate is around 16 (2x) for 8 frames, around 128 (8x) for 16 frames, around 256 (8x) for 32 frames, and over 640 (10x) for 64 frames.

We observed that the relation of the approximated optimal rate to the number of frames is not linear (faster

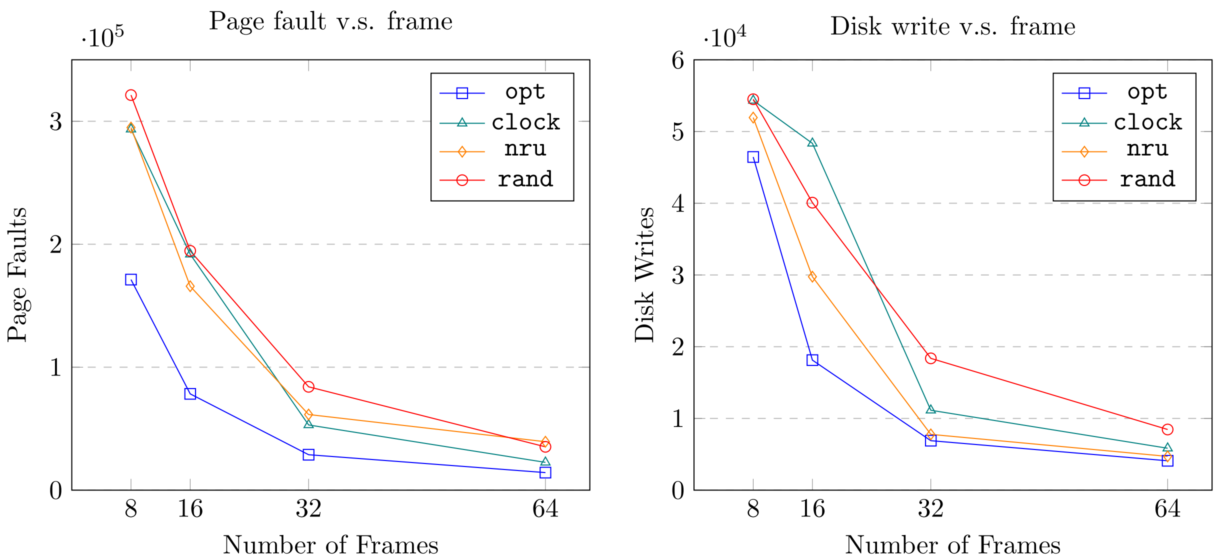

Figure 2: `swim.trace` page faults and disk writes

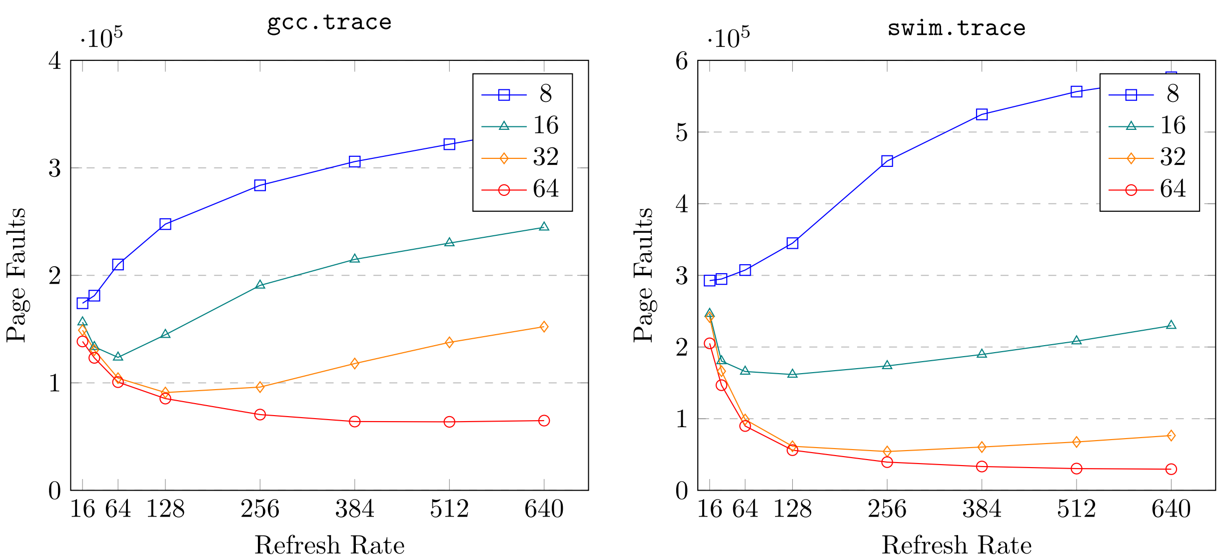

Figure 3: Page fault vs. refresh rate

Frames	Rate	Page Faults	Disk Writes
8	16	174046	18193
	32	181092	19333
	64	210090	18586
	128	247677	17251
	256	283767	16842
	384	305836	17254
	512	321980	18678
	640	337768	20410
16	16	156441	13412
	32	133487	11998
	64	123710	9711
	128	144753	13679
	256	190646	13553
	384	214867	13231
	512	230018	12867
	640	244660	12921
32	16	148978	11273
	32	130396	10807
	64	104364	9711
	128	90990	9289
	256	96056	11106
	384	117962	11272
	512	137667	11215
	640	152277	10966
64	16	138512	9801
	32	123256	10193
	64	100676	9378
	128	85372	7951
	256	70481	6705
	384	64012	6718
	512	63732	7414
	640	64874	8024

Metrics for `gcc.trace`

Frames	Rate	Page Faults	Disk Writes
8	16	292713	51575
	32	294885	51943
	64	307397	49286
	128	344913	35455
	256	459516	25246
	384	524616	21308
	512	556403	20970
	640	576372	20712
16	16	246505	28155
	32	180457	21150
	64	165840	29760
	128	161641	23756
	256	173624	17693
	384	189591	14815
	512	208184	13737
	640	229762	12693
32	16	241775	27299
	32	166296	16113
	64	98822	10759
	128	61518	7778
	256	54201	8250
	384	60449	8346
	512	67521	8143
	640	76486	7948
64	16	205105	20227
	32	146737	14302
	64	89813	9489
	128	56082	6605
	256	39402	4707
	384	33259	4364
	512	30364	4435
	640	29552	4590

Metrics for `swim.trace`

Table 2: Performance statistics across refresh rates

than linear in both trace files) and varies from process to process. Based on the metrics, we propose setting the refresh rate to 4 times of the number of frames n as a linear approximation for only $n < 64$. This choice is also verified by our observations in Section 1.3.2 where the refresh rate is set to $4n$.

1.4 Conclusion

After comparing and analyzing the performance of the four (three practically implementable) page replacements algorithms, we conclude that the second chance clock algorithm is the most preferable one for modern Operating Systems. The clock algorithm consistently causes relative low number of page faults compared to the random algorithm and it does not require additional parameters such as the refresh rate for the NRU algorithm. A major downside of the NRU algorithm, other than its ineffectiveness in maintaining historical usage data, is the difficulty to determine the optimal refresh rate. As we have observed in Section 1.3.3, it also depends on the execution of the process itself. This leads to the consideration to have variable refresh rate per execution, which further complicates the implementation. Finally, although the NRU algorithm generally yields fewer (sometimes nearly optimal) disk writes compared to other ones, as we have acknowledged when designing the NRU algorithm (that the D bit is less significant than the R bit), the number of page faults has a greater impact on the overall performance than the number of disk writes, for which the NRU algorithm is suboptimal to the clock algorithm.