Memory Systems a Report

For DV1628/DV1629 Lab 2

Alexander Järnström | 010916-6298 | 08/10/24

aljr21@student.bth.se

Table of Contents

Home Assignment 3

Struct link has the size of 4’294’967’296 bytes (4 GB).

Each write is 1’073’741’824 bytes (1GiB).

Home Assignment 4

VMSTAT

- **SWPD**: Amount of virtual memory used.
- **BO**: IO read.
- **BI**: IO Write.
- **SI**: Swap ins.
- **SO**: Swap outs.

TOP

- **TIME+**: Amount of CPU time used.
- **%CPU**: Current CPU usage in percent.

Task 1

Test 1

- **Memory usage**: ~4GB, which matches the answer in *Home Assignment 3*.
- **CPU usage**:
  - **Single core**: 100%
  - **Whole CPU**: 6.2%

Test1 is CPU bound.

Test 2

- **Written blocks**: 40
- **Memory usage**: ~1.1 GB
- **CPU usage**:
  - **Single core**: 23%
  - **Whole CPU**: 1.8%

The test was run four times which gave:

$$4 * 1.1 = 4.4 \text{ GB}$$

and is close to the written 40 blocks, this show *Test2* is IO bound.

Task 2

Question 1

Run1 executes the commands sequentially while *Run2* executes the two sequential parts at the same time.

Question 2

Run1 took one minute and 0.76 seconds with a CPU usage (whole CPU) of 6.2% on the first part and 0.9% on the second part.

Question 3

Run2 took 2.249 seconds which is incorrect, it leaves out the time spent in the background processes. It used 7.1% of the CPU initially, when *Test2* was done it used 6.2%.

Question 4

Run2, due to *Test1* using a lot of CPU while *Test2* is using a lot of IO, they are not overlapping which means they are able to run simultaneously without hurting one another, while *Run1* is taking all the time it can get.

Task 3

Used modulo first when getting the address, which gave the wrong results, changed it to integer division.

Task 4

Table 1: Number of page faults for mp3d.mem when using FIFO as page replacement policy.

Page Size	***Number of pages***							
	1	**2**	**4**	**8**	**16**	**32**	**64**	**128**
128	55421	22741	13606	6810	3121	1503	1097	877
256	54357	20395	11940	4845	1645	939	669	478
512	52577	16188	9458	2372	999	629	417	239
1024	51804	15393	8362	1330	687	409	193	99

Table 2: umber of page faults for mult.mem when using FIFO as page replacement policy

Page Size	***Number of pages***							
	1	**2**	**4**	**8**	**16**	**32**	**64**	**128**
128	45790	22303	18034	1603	970	249	67	67
256	45725	22260	18012	1529	900	223	61	61
512	38246	16858	2900	1130	489	210	59	59
1024	38245	16855	2890	1124	479	204	57	57

Task 5

Question 1

The amount of page faults slowly improves due to the page having a greater chance of holding the wished address, when it grew bigger.

Question 2

With a bigger page table, the chances of having a page grows, which in turn gives fewer page faults.

Question 3

It depends on how *lucky* we get with the memory requests. If it just so happens the page size cover a given portion, no page fault would occur, while other times a split would happen and a second page would be needed.

Question 4

The number of page faults drastically decrease between sizes **256** and **512**, which might be because the running program is constantly asking for memory outside the scope of pages with size 256, but when the size is increased to 512 we suddenly cover the whole area.

Question 5

It stops improving at **64** and **128**, it is because of the initial page faults. Where for count **64** there are 64 initial page faults before the table is full and then there are 3 actual misses, where **128** did not fill up completely.

Task 6

Was challenging finding a way to keep track of when pages where called, ended up creating a struct with the needed data.

Task 7

Table 3: umber of page faults for mp3d.mem when using LRU as replacement policy.

Page Size	***Number of pages***							
	1	**2**	**4**	**8**	**16**	**32**	**64**	**128**
128	55421	16973	11000	6536	1907	995	905	796
256	54357	14947	9218	3811	794	684	577	417
512	52577	11432	6828	1617	603	503	362	206
1024	51804	10448	5605	758	472	351	167	99

Task 8

Question 1

Due to Least Recently Used (LRU) keeping track on which pages have not been touched for a while, the chances for the algorithm to make a better choice increase, therefore LRU does a better job than FIFO at keeping the right pages in memory.

Question 2

If the least recently used pages happens to also be the oldest ones kept in the table, they would be removed in both algorithms, making them as efficient in those specific sequences of memory calls. Another reason could be the size of the pages or the amount of them, if there only is one, the kind of algorithm does not matter, there are no choices to be made. Or if the amount is great enough they will be able to cover all calls except the initial page faults.

Task 9

Finding a convenient way to update when the page call would be referenced next was a hassle, but simply solved with several linear search algorithms.

Task 10

Table 4: Number of page faults for mp3d.mem when using Optimal (Bélády's algorithm) as replacement policy

Page Size	***Number of pages***							
	1	**2**	**4**	**8**	**16**	**32**	**64**	**128**
128	55421	15856	8417	3656	1092	824	692	558
256	54357	14168	6431	1919	652	517	395	295
512	52577	11322	4191	920	470	340	228	173
1024	51804	10389	3367	496	339	213	107	99

Task 11

Question 1

It is easier to know which pages to throw away when there is access to future memory calls. The pages with calls the furthest away are not as necessary at the moment as the closest ones.

Question 1

Due to the algorithms dependence on future information, there is no way to implement it, the OS does not know which calls are needed at any future time.

Question 1

They have the same fault count when the page amount is one and when the page count is 128 with size 1024, when there only is one page, the algorithm does not matter. Either the next call is in the page or not there is no choice, and when there is an abundance of pages, with great sizes, all the memory calls are covered, so only the initial faults appear.

Test Code

```
#!/bin/bash

page_sizes=(128 256 512 1024)
n_pages=(1 2 4 8 16 32 64 128)

for p in ${page_sizes[@]}; do
  for n in ${n_pages[@]}; do
    ./fifo $n $p ../test/mp3d.mem
  done
done
```

Code 1: Code 1: test run FIFO with mp3d.mem

```
#!/bin/bash

page_sizes=(128 256 512 1024)
n_pages=(1 2 4 8 16 32 64 128)

for p in ${page_sizes[@]}; do
  for n in ${n_pages[@]}; do
    ./fifo $n $p ../test/mult.mem
  done
done
```

Code 2: Code 1: test run FIFO with mult.mem

```
#!/bin/bash

page_sizes=(128 256 512 1024)
n_pages=(1 2 4 8 16 32 64 128)

for p in ${page_sizes[@]}; do
  for n in ${n_pages[@]}; do
    ./lru $n $p ../test/mp3d.mem
  done
done
```

Code 3: Code 1: test run LRU with mp3d.mem

```
#!/bin/bash

page_sizes=(128 256 512 1024)
n_pages=(1 2 4 8 16 32 64 128)

for p in ${page_sizes[@]}; do
  for n in ${n_pages[@]}; do
    ./optimal $n $p ../test/mp3d.mem
  done
done
```

Code 4: Code 1: test run Optimal with mp3d.mem