Labb 2, DV1629

Marius Stokkedal
Matk20@student.bth.se
DVAMI20

Home Assignment 3

How much dynamic memory is allocated in the test1 program (i.e., how large is struct link)? The struct is 4*(32*1024)^2=4294967296 bytes which is exactly 4GB

How much data is written to the temporary file /tmp/file1.txt in each iteration of the program test2? SIZE*SIZE = (16*1024)^2 = 268435456 elemnts. Each element is an int which is 4 bytes i.e 268435456*4=1073741824 bytes which is exactly 1GB

Home Assignment 4

In which output columns of vmstat can you find the amount of memory a process uses, the number of swap ins and outs, and the number of I/O blocks read and written? The memory columns, swap columns (si and so) and the io columns (bi and bo).

Where in the output from top do you find how much cpu time and cpu utilization a process is using? Time+ and %CPU

Task 1

Execute the test program test1. How much memory does the program use? Does it correspond to your answer in Home Assignment 3? Using the top-command I can see that it uses 419708 which equals about 4.2GB. This roughly matches the prediction from HA3 which was 4GB

What is the cpu utilization when executing the test1 program? Which type of program is test1? By using the top-command again it is possible to see the CPU usage. When running test1.c the CPU usage is at 100%. This means that test1.c is a CPU bound program i.e the program is limited by the computing speed of the CPU.

Execute the test program test2. How much memory does the program use? How many blocks are written out by the program? How does it correspond to your answer in Home Assignment 3? Test2 uses a bit over 1GB of memory which also matches the prediction from HA3. When testing the result varies between about 120 000 and 245 000 blocks

What is the cpu utilization when executing the test2 program? Which type of program can we consider test2 to be? The CPU utilization when executing test2 is about 40%, therefore test2 is a I/O bound program i.e the program is limited by the writing speed to the file.

Task 2

What is the difference between them in terms of how they execute the test programs? Run1 will run the programs sequentially while Run2 will run them simultaneously.

Execute the script run1 and measure the execution time. Study the cpu utilization using top during the execution. How long time did it take to execute the script and how did the cpu utilization vary? It took about 52 seconds to execute run1. The CPU utilization varied between 100% and about 40%.

Execute the script run2 and measure the execution time. Study the cpu utilization using top during the execution. How long time did it take to execute the script and how did the cpu utilization vary? It took about 44 seconds to execute run2. The CPU utilization varied between 50% and 60%.

Which of the two cases executed fastest? Run2 executed fastest

In both cases, the same mount of work was done. I which case was the system best utilized and why? Run2 runs the programs in parallel instead of sequentially.

Task 3

Code is commentated. When a needed page isn't found in the page table the one which gets thrown out is the oldest without regard of usage.

Task 4

Results for task 4 can be found in the *Tables* section

Task 5

What is happening when we keep the number of pages constant and increase the page size? Explain why! The page faults decrease by a little bit because the chance of addresses being in the same page increases.

What is happening when we keep the page size constant and increase the number of pages? Explain why! The page faults decrease much faster, this is because an address can be loaded onto other pages if they are not already in the table.

If we double the page size and halve the number of pages, the number of page faults sometimes decrease and some-times increase. What can be the reason for that? It happens when the number of pages is 32 and above. Before that the page faults decrease when the number of pages increase. The page faults behave like this because when increasing the size of the pages in the page table the range of addresses increases. When loading a new address the likelihood of its page already being in the page table increases and therefore decreases the page faults since a new page doesn't have to be loaded.

Focus now on the results in Table 2 (matmul). At some point decreases the number page faults very drastically. What memory size does that correspond to? Why does the number of page faults decrease so drastically at that point? The big difference is for memory size 2048 (4*512 or 8*256) When increasing the size of the pages or the amount of pages in the table a larger part of the working set is covered i.e. there is a larger chance that the page which is needed already exists in the page table and therefore avoiding a page fault.

At some point the number of page faults does not decrease anymore when we increase the number of pages. When and why do you think that happens? The page size is big enough for all the data to fit so the amount of pages won't make a difference.

Task 6

Also commented code. Every time we get a page hit that page is moved to the end of the queue and when a page fault happens the page first in line, the least recently used one, is removed.

Task 7

See table 3 in *tables* section for result

Task 8

Which of the page replacement policies FIFO and LRU seems to give the lowest number of page faults? Explain why! LRU is performing better in most cases and this is because it takes in account how often pages in the page table are used (if a page is used it is put at end of the line).

In some of the cases, the number of page faults are the same for both FIFO and LRU. Which are these cases? Why is the number of page faults equal for FIFO and LRU in those cases? Explain why! This occurs in two cases. When the page size is 1, because there is only one page in the table the replacement algorithm won't make any difference, and when the page size is 1024 and the number of pages is 128. This is because both algorithms have reached the best case. The amount of memory and pages that is used makes it less important for the algorithm to be efficient.

Task 9

Also commented code. All memory addresses is loaded in before the algorithm is loaded in to an array. When a page fault occurs every page in the page table is checked for when it is needed the next time. THe one which gets thrown out is the one needed the furthest away in time.

Task 10

See table 4 in *tables* section for result

Task 11

As expected, the Optimal policy gives the lowest number of page faults. Explain why! Because it looks which pages will be used in the future when throwing away a page, the one which gets thrown is the one which will be used furthest away in time or never will be used again.

Optimal is considered to be impossible to use in practice. Explain why! Because it looks forward to knowing which page to throw away, which would mean it would need to look into the future.

Does FIFO and/or LRU have the same number of page faults as Optimal for some combination(s) of page size and number of pages? If so, for which combination(s) and why? Yes, firstly, as discussed earlier, when the number of pages the page faults will be the same regardless of page size since there is only one page and it can't be replaced in a more optimal way. The second case is when the number of pages is 128 and the page size is 1024. The page fault is below the amount of pages (to fill the page table with new pages will generate a page fault for every new page), this means that all the addresses in the data have been loaded into the table.

Tables

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

Page Size	**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: Number of page faults for mult.mem when using FIFO as page replacement policy

Page Size	**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

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

Page Size	**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	415
512	52577	11432	6828	1617	602	503	362	206
1024	51805	10448	5605	757	472	351	167	99

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

Page Size	**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	51805	10389	3367	496	339	213	107	99