

KIT104 ICT Architecture and Operating Systems

Tutorial Week 13 – UNIX Virtual Memory Page Faults

Introduction

In this tutorial you are going to set up simulated experiments to determine the effects on the **page fault rates** by changes in these parameters: page size, memory size, and page replacement algorithm.

You are not required to understand all aspects of the C programs provided in this tutorial. It should be sufficient to be aware of the nature of the programs and their functional descriptions provided here.

A	Create a new directory called <code>wk13</code> under your <code>kit104</code> directory.
B	Copy all the C programs (totally 5) from the following directory on your UNIX server to your newly-created directory <code>wk13</code> (you should be able to use a single command line to do so): <code>/units/kit104/tutorials/wk13</code>
C	Compile these programs into files with names matching the names of the C programs. Thus, <code>LRU.c</code> is compiled to generate executable <code>LRU</code> . Here are the commands: <pre>\$ gcc -o nameSorter nameSorter.c \$ gcc -o pageRefGen pageRefGen.c \$ gcc -o FIFO FIFO.c \$ gcc -o LRU LRU.c \$ gcc -o OPT OPT.c</pre>

Task1: nameSorter.c

The program `nameSorter.c` is our generator program. It randomly generates a lot of names. The version given can generate up to 5000 names. It then sorts these names using a quick sorting algorithm. You can print unsorted names or sorted list of names on your screen. What you want this program to do is specified through command line arguments -- one or more character strings placed in the command line after the command.

D	Read this program (you should at least read the comment lines to get to know what each function does). Here is how you run the program to create 500 names and list them in an unsorted order (I hope you compiled the program as advised earlier). <code>./nameSorter 500 listnames</code>
---	--

	If you check the source code you'd see that <code>listnames</code> is one of the arguments specified by this program.
E	You can not see 500 names on your screen, can you? Use the following command line: <code>\$/nameSorter 500 listnames more</code>
F	To create 1000 names and list them in a sorted order you use the following command: <code>\$/nameSorter 1000 listsorted</code> If you check the source code you'd see that <code>listsorted</code> is also one of the arguments specified by this program.
G	Two important data structures in the program <code>nameSorter.c</code> are char arrays: <code>list</code> and <code>names</code> . <i>During a sorting exercise, data stored in these arrays are accessed for reading their content and for writing new values into them.</i> The program uses C pointer arithmetic to list these accesses on your screen. The output of the following command line will be different: <code>\$/nameSorter 500 listaccess</code> (Similar to <code>listnames</code> and <code>listsorted</code> , <code>listaccess</code> is also one of the arguments specified by the program.)
H	What you see on your screen now are the (virtual) memory addresses, or logical addresses, of the process being accessed during the sorting. The output uses letter R to indicate a read access; W to indicate a write access; and F 000 to indicate finish.
I	Find out how many read accesses are there in the list generated by the previous command line. How many are writes?

Task 2: `pageRefGen.c`

File `pageRefGen.c` is a C program from which you generate `pageRefGen`. This filter can read the list of memory accesses generated by the `nameSorter` and convert them into references to the page numbers. For example, if we know that address 1000 was read and each page is 256-byte long, then page number $1000/256 = 3$ was accessed to read data from the address 1000 (this means that the address 1000 is within page 3). The command line argument for the program specifies the size (number of bytes) of a page.

J	To test the program run the following command: <code>\$/pageRefGen 256</code> R 1000 (user input) R 3 (response of the program)
---	--

	<p>To continue the testing enter the following lines (press Enter at the end of each line):</p> <pre>R 9987 R 6740 W 1000 W 7609</pre> <p>To quit, enter F 0.</p>
K	<p>Now, we run the program by feeding it input from another program through a pipe.</p> <pre>\$./nameSorter 500 listaccess ./pageRefGen 256</pre> <p>The output of this command line can be considered as a long memory reference string. From the lectures you know that we can evaluate a page replacement algorithm by running it on a particular memory reference string and computing the number of page faults – this is what you are going to do next.</p>

Task 3: Virtual Memory Page Faults

We provide three programs **FIFO.c**, **LRU.c**, and **OPT.c** to implement page replacement algorithms for simulated virtual memory systems.

The following is a brief summary of these algorithms.

FIFO (=First In First Out) policy is very simple to implement. It assumes that a page that came in the main memory first will be the first to go out when we need space for some fresh pages.

LRU (=Least Recently Used) policy is based on the following premises: If a page has not been used for sometime then it will not find a use in the future too. So the page that is most likely to be never used in the future is the one which has not been used for longest continuous period ending at the present time. Usually finding such a page is a moderately difficult process.

OPT (Optimal) policy is computationally very expensive but provides best results.

What we will do here is to test the page faults that result from the use of these algorithms. A page fault happens when accessed page is not found in the memory and causes the page to be read from the disk. In addition, some page faults may cause a disk write of the modified data to precede the disk read (i.e., write a dirty page to disk first).

FIFO, **LRU**, and **OPT** need one command line argument to know the size of the simulated memory (number of initially available frames). We also need this to determine the number of page faults.

L	<p>Run the following commands and briefly explain which number (in the output) represents the number of page faults, and which number represents the number of dirty pages?</p> <pre> \$./nameSorter 500 listaccess ./pageRefGen 64 ./LRU 64 \$./nameSorter 500 listaccess ./pageRefGen 64 ./LRU 32 </pre>
M	<p>Run the following commands to compare the number of page faults using the three page replacement algorithms.</p> <pre> \$./nameSorter 500 listaccess ./pageRefGen 64 ./FIFO 64 \$./nameSorter 500 listaccess ./pageRefGen 64 ./LRU 64 \$./nameSorter 500 listaccess ./pageRefGen 64 ./OPT 64 \$./nameSorter 500 listaccess ./pageRefGen 32 ./FIFO 64 \$./nameSorter 500 listaccess ./pageRefGen 32 ./LRU 64 \$./nameSorter 500 listaccess ./pageRefGen 32 ./OPT 64 \$./nameSorter 500 listaccess ./pageRefGen 64 ./FIFO 32 \$./nameSorter 500 listaccess ./pageRefGen 64 ./LRU 32 \$./nameSorter 500 listaccess ./pageRefGen 64 ./OPT 32 \$./nameSorter 500 listaccess ./pageRefGen 32 ./FIFO 32 \$./nameSorter 500 listaccess ./pageRefGen 32 ./LRU 32 \$./nameSorter 500 listaccess ./pageRefGen 32 ./OPT 32 </pre>

(The End)