Operating Systems – spring 2022
Assignment 6

Instructor: Hans P. Reiser

Submission Deadline: Monday, February 28th, 2022 – 23:59

A new assignment will be published every week, right after (or a bit before) the last one was due. It must be completed before its submission deadline. (Hard deadlines, no extension.)

T-Questions are theory homework assignments. **The answers to the assignments must be uploaded to Canvas (in the quiz).**

P-Questions are programming assignments. Download the provided template from Canvas. Do not fiddle with the compiler flags. Submission instructions can also be found in first assignment.

The topic of this assignment is a deeper dive into page fault handling and page replacement.

T-Question 6.1: Paging

a. How can shared memory between two processes A and B be realized an the page table level? **1 T-pt**

b. How does the page protection in the PTE for a copy-on-write page need to be configured? Explain your answer! **1 T-pt**

T-Question 6.2: Page Replacement

a. A page table may contain named entries (memory mapped files, such as code of an executable program) and anonymous mappings (page not backed by a file, such as the heap and stack of a program). Explain why (and how) a page fault handler has to handle these two cases differently when it needs to (1) evict a page from physical memory and (2) load a page into physical memory. **2 T-pt**

b. Consider a system with a total of 4 page frames and an application accessing memory according to this reference string of virtual page numbers (VPNs): **4 1 2 0 3 5**. Complete the mapping table for a process at different times (t_0: initial mapping, t_1: mapping after accessing VPN 4, etc.) if **clock page replacement** is used. Assume that the circular buffer of the clock is in ascending order (i.e., frame 0, 1, 2, 3), that the clock hand at t_0 is positioned at **frame 0** and that the reference bit for **page 0** is set, and cleared for the other pages. **3 T-pt**

frame	$VPN(t_0)$	$VPN(t_1)$	$VPN(t_2)$	$VPN(t_3)$	$VPN(t_4)$	$VPN(t_5)$	$VPN(t_6)$
0	3						
1	1						
2	0						
3	6						

c. Page faults for different algorithms **3 T-pt**

Note: The access sequence for this question was generated by the following command (see P part): `paging-policy.py -s 6 -n 15 -m 5`

Consider the page access sequence (reference string):

```
3 4 2 1 0 3 2 3 1 3 1 4 3 2 2
```

For each of the three policies FIFO, LRU and OPT, determine which of these access are page faults ("cache misses") in a system with a total of four physical frames ("cache size")! What is the total number of page faults?

P-Question 6.1: Page Replacement Simulation

For this assignment (only), there is no C programming required. Instead you will be doing experiments with page replacement strategies using the simulator from the OSTEP book (see `https://github.com/xxyzz/ostep-hw/blob/master/22/paging-policy.py` or Canvas for a version with minor adjustments for Python 3). Please note the terminology used in the script (see also book chapter 22): The main memory of a process is seens as a "cache" of the virtual memory of a process. Consequently, the number of physical pages mapped in an application's virtual address space is called the "cache size".

a. Worst-case reference strings. Find a single, arbitrary access pattern with 20 elements that has an worst-case performance on all of these scheduling policies: FIFO, LRU, CLOCK and OPT. **1 P-pt**

   You can simulate the execution of an arbitrary access pattern using the following command: (example: access pattern 1,2,3,1,2,3,1,2,3 on a system with cache size 4 and policy FIFO):

   ```
   ./paging-policy.py -a 1,2,3,1,2,3,1,2,3 -C 4 -p FIFO -c
   ```

b. More worst-case reference strings. For each of the three policies FIFO, LRU and MRU, find one access pattern that uses at most 5 virtual memory pages and has a worst-case beheviour (maximizing page faults) on a system with four physical frames. **3 P-pt**

   For the submission of part (a) and (b), put your worst-case reference strings into the file `def-worstcase.sh`. You can run all simulations using `run-worstcase.sh`

c. Paging policy silmulation with real workloads. Run the following command on skel to obtain a real memory access trace: **2 P-pt**
   `valgrind --tool=lackey --trace-mem=yes ls -la`
   Implement a Python script `transform-lackey.py` that transforms the lackey output in a list of pages. The script shall read the lackey output on standard input and write virtual page numbers on standard output. (Note: the valgrind output contains lines with type of access (I:code/L:load/S:store/M:modify), virtual address of the access, and access size. You may discard the type and size of the access, and simply map the virtual address to the virtual frame)

d. Page replacement simulation Use the trace from part (c) and the simulation tool to calculate the page hit rate for several sizes of the physical memory, for each of the policies FIFO, LRU, RAND and OPT. Plot a graph with the results: X-axis: number of pages in physical memory (1..total number of accessed blocks), Y-axis: page hit rate. **4 P-pt**
   Submission: a single file `simresult.png` with your results (all four policies plotted in a single diagram).

Total:
10 T-pt
10 P-pt