HÁSKÓLI ÍSLANDS
Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild

TÖL401G: Stýrikerfi / Operating Systems · Vormisseri 2022

Assignments 21–22 · Due 5.4.2022, 13:00

Assignment 21

Consider the following page reference string of a program referencing four pages:

Reference number	1.	2.	3.	4.	5.	6.	7.	8.
Page reference string	0	1	3	1	2	0	1	2
Read or write access (R=Read / W=Write)	R	R	W	W	R	R	R	W

Assume that the program is executed in a system with 3 frames that are initially empty (and thus even their initial allocation leads to page faults): What is the contents of the frames after each access and how many page faults occur using the page replacement policies below?

1. *First-In-First-Out* (FIFO: either using Clock-like implementation or using real FIFO queue that copies pages from frame to frame),
2. *Second-Chance/Clock*
3. *Enhanced-Second-Chance* (also called NRU),
   Note: A periodic reset of the "referenced" bit of all page table entries takes place just immediately after the 3rd and the 6th reference.
4. *Least-Recently-Used* (LRU),
5. *Optimal* (OPT).

Note: It is easiest to use a spreadsheet for filling in the solutions:
If you have a Google account, you can work online on a copy of a template: `https://docs.google.com/spreadsheets/d/1smlBMRc6AnzpkH_YQ9XVxajCgAqf-kMtxZYaqt6jDf4/copy`
If you have no Google account, you view and download a copy a template from here: `https://docs.google.com/spreadsheets/d/1smlBMRc6AnzpkH_YQ9XVxajCgAqf-kMtxZYaqt6jDf4/edit?usp=sharing`

Assignment 22

Consider a system that offers virtual memory using demand paging: The page or respectively frame size is 1 KB, the size of the physical RAM is just 1 frame. The used page replacement policy is *Least-Recently-Used* (LRU). In this system, the following C-like program[1] is executed:

```
int i,j;
int data[128][128];

for (j = 0; j < 128; j++) {
  for (i = 0; i < 128; i++) {
    data[i][j] = 0;
  }
}
```

Assume that the instructions of this program are located in ROM and that the variables `i` and `j` are stored in a CPU register (i.e. neither the program instructions nor the variables `i` and `j` require a frame of the RAM, hence the available single frame can be solely used for storing the array). Each `int` element of the array occupies 4 bytes and the array starts at a page boundary.

1. How many pages are occupied by the array `data` in logical memory (=virtual memory)?
2. How many page faults occur if the program is executed and the frame is initially empty?
3. How would you have to modify the program to obtain the same final array values with a minimal number of page faults?
4. What is the minimal number of page faults required by a program that sets the same final array values and is executed in the same environment?

Preparation

Read chapter 9 as preparation for this week assignments.

Examples from the chapter are explained in videos that are available in Canvas via the tab "Panopto´´.

Report via Piazza any questions that you have, so that we can clarify this during remote class or directly via Piazza.

[1] In C, declaring an array (such as the line `int data[128][128];`) does not access the array elements – it rather tells the compiler to reserve the right amount of space in memory without initialising it. Furthermore, C stores multi-dimensional arrays in *Row-major order*, i.e. row by row. Thus, the resulting storage order of the above array elements in consecutive memory addresses is: `data[0][0],data[0][1],...,data[0][127],data[1][0], data[1][1],...,data[1][127],...,data[127][0],data[127][1],...,data[127][127]`.