Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CO20-320202
Date: 2019-10-28
Due: 2019-11-04

OS 2019 Problem Sheet #4

Problem 4.1: *page replacement algorithms* (1+1+1+1+1+1 = 6 points)

An operating system uses demand paging to implement virtual memory. A user-space process uses four pages, numbered 1 to 4. The pages are accessed in the following order (reference string): 1, 2, 3, 4, 1, 1, 4, 2, 1, 2. None of the pages used by the user-space process are resident when the process starts and the frames allocated to the process are unused. Fill out the following tables by writing page numbers into the cells. Each number indicates which page a frame holds after accessing the page indicated by the reference string. Write down the number of page faults for each scenario.

a) Show how the pages are mapped to two frames using the First-In-First-Out (FIFO) page replacement algorithm.

reference string	1	2	3	4	1	1	4	2	1	2
frame 0										
frame 1										

b) Show how the pages are mapped to three frames using the First-In-First-Out (FIFO) page replacement algorithm.

reference string	1	2	3	4	1	1	4	2	1	2
frame 0										
frame 1										
frame 2										

c) Show how the pages are mapped to two frames using Belady's Optimal (BO) page replacement algorithm.

reference string	1	2	3	4	1	1	4	2	1	2
frame 0										
frame 1										

d) Show how the pages are mapped to three frames using Belady's Optimal (BO) page replacement algorithm.

reference string	1	2	3	4	1	1	4	2	1	2
frame 0										
frame 1										
frame 2										

e) Show how the pages are mapped to two frames using the Least Recently Used (LRU) page replacement algorithm.

reference string	1	2	3	4	1	1	4	2	1	2
frame 0										
frame 1										

f) Show how the pages are mapped to three frames using the Least Recently Used (LRU) page replacement algorithm.

reference string	1	2	3	4	1	1	4	2	1	2
frame 0										
frame 1										
frame 2										

Hint: Do not move pages between frames except when necessary.

Problem 4.2: *dynamic loadable object files* (1+2+1 = 4 points)

POSIX.1-2001 defines the functions `dlopen()`, `dlclose()`, `dlsym()`, and `dlerror()` to dynamically link object files into a program. This API can be used to split a complex program into a small core that is extended as needed by loading additional object code. Read the `dlopen(3)` man page for details. The Linux version of the man page includes an example.

The source code repository of the operating systems course contains an implementation of `cat++`, a cat program that can dynamically load transformations that are applied to lines before they are printed. Compile the program, study the source code, and answer the following questions.

a) What does the following shell command produce?

```
echo "hello world" | \
        ./cat++ -l ./librot13.so -l ./libupper.so -l ./librot13.so
```

Explain what the transformations do and in which order they are applied.

b) How many memory segments are added to the process when one of the libraries implementing a transform is loaded? What happens if a library is loaded multiple times, like shown in the example above? Hint: Use `pmap` to obtain the memory map of a running process.

c) Does the current version of the `cat++` program allow you to implement transforms that for example emulate `cat -n` or `wc`? Explain why or why not.