Operating Systems 2016/17
Assignment 11

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

Submission Deadline: Monday, January 23th, 2017 – 23:59

A new assignment will be published every week, right after the last one was due. It must be completed before its submission deadline.

The assignments must be filled out online in ILIAS. Handwritten solutions are no longer accepted. You will find the online version for each assignment in your tutorial's directory. **P-Questions** are programming assignments. Download the provided template from ILIAS. Do not fiddle with the compiler flags. Submission instructions can be found on the first assignment.

In this assignment you will dive deeper into page fault handling and page replacement.

T-Question 11.1: Paging

Consider a system that translates virtual addresses to physical addresses using two-level page tables in hardware. Every page table comprises 1024 entries, with each entry having a size of 4 bytes and providing read/write page protection as presented in the lecture. The page size is 4096 bytes. The system neither possesses a cache nor a TLB.

a. How many memory accesses are necessary to read a contiguous buffer of 8 MiB, starting at offset 0, reading 4 bytes at a time. **1 T-pt**

b. How many memory accesses are required if a TLB is added to the system? **1 T-pt**

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

d. 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 11.2: Page Replacement

a. Consider a system with 4 page frames. Complete the mapping table for a process that accesses pages in the given order if clock page replacement is used. Assume the circular buffer of the clock to be in ascending order (i.e., frame 0, 1, 2, 3), the clock hand to be positioned at frame 0 and the reference bit for page 0 to be set. **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						

Reference string for virtual page numbers (VPN): **4 1 2 0 3 5**

b. What difficulty do you see when implementing the clock algorithm for systems that allow shared memory? **1 T-pt**

P-Question 11.1: Page Fault Handling

Download the template **p1** for this assignment from ILIAS. You may only modify and upload the file `page_faults.c`.

Consider the same system and page table structure as used in the previous programming question (P9.1). See the assignment 9 for more details.

In this question you will extend the software MMU for x86 paging from the last assignment with a page fault handler. For simplicity reasons, the page fault handler will only be called if a PTE is marked as invalid and not if the access check in the MMU failed.

The template already provides structures to represent the virtual memory areas (VMAs) of an exemplary address space as well as utility functions to find the right VMA for a virtual address and to simulate the allocation of zero-filled and file-backed frames. You will find the VMA definitions in the `_vmas` variable.

For all questions you may assume that frames are always mapped at most once (i.e., no shared memory).

a. Write a function that swaps out the page specified by the given virtual base address. Your function should fulfill the following requirements: **3 P-pt**

- Retrieves the page table entry for the supplied virtual base address.
- Calls `_storeOnDisk()` to simulate the swap operation and get the resulting offset of the page on disk
- Stores the disk offset in the PTE and marks the page as swapped out by setting the appropriate bit via `PAGE_SWAPPED_MASK`. Updates the other bits as needed, however, preserves the access bits defined in `PRESERVED_BITS_ON_SWAP`.
- Returns 0 on success, -1 otherwise.

```
int swapOut(uint32_t virtualBase);
```

b. Implement the page fault handler that prepares and maps a frame depending on the accessed page (and the corresponding VMA). Your implementation should fulfill the following requirements: **4 P-pt**

- Checks if the accessed page has been swapped out. In that case, simulates the load from the swap by calling `_loadFromDisk()` and updates the PTE accordingly.
- Otherwise, checks what type of VMA the accessed page belongs to via `_getVMA()` and prepares and maps a new frame with `_getFilePage()` or `_getZeroedPage()`, respectively. You can use `mapPage()` to establish the mapping. Be sure to specify the correct permissions.
- Returns 1 on success, -1 on any error, and 0 if the accessed address is not part of a VMA and thus invalid.

```
int handlePageFault(uint32_t virtualBase, uint32_t pte);
```

Total:
8 T-pt
7 P-pt