Operating systems

Fourth laboratory exercise

March 10, 2023

Contents

1 Introduction

The goal of this laboratory exercise is to study virtual memory systems and address space isolation on modern hardware. Your main task is to implement a simulation of a simple computer system using virtual memory.

2 Tasks

2.1 Paging simulation (4 boda)

As Vaš je zadatak ostvariti simulaciju rada više procesa u sustavu koji koristi mehanizam straničenja na zahtjev pomoću arhitekture prikazane na slici 1.

The simulated system consists of N processes, a hard drive, an array of M frames and a page table for each simulated process. Your simulation should follow the workflow the specified in Algorithm 1 and must allow the user to specify the number of frames (M), and the number of processes (N). Your implementation should also have the following data structures:

- `hard_drive[N]` - A simulated hard drive used for storing frame contents,
- `frames[M]` - Simulated physical memory consisting of M 64-byte frames,
- `page_tables[N]` - Page tables for each of the N simulated processes.

Algorithm 1: Simulation pseudocode.

```
for i = 1 to N do
    create process i;
    initialize page table for process i;
end
t ← 0;
while
    for each process p do
        x ← random virutal address;
        i ← read_contents(p, x);
        i ← i + 1;
        write_value(p, x, i);
        t ← t + 1;
        sleep;
    end
```

Algorithm 2: Helper function pseudocode.

```
1  Funkcija read_contents(p, x)
2      y ← translate_address(p, x);
3      i ← contents at y;
5      return i;
1  Funkcija write_value(p, x, i)
2      y ← translate_address(p, x);
3      write i to address y;
1  Funkcija translate_address(p, x)
2      find process p page table entry for address x;
3      if adress x not valid then
4          print page fault;
5          find and allocate frame;
6          load frame content from hard drive;
7          update page table for process p;
8      end
9      print address x and the address of its corresponding frame and
         page table entry;
11     return physical address;
```

The address space of a process must be accessed using the `read_contents` and `write_value` helper functions. Any solution that does not adhere to this rule **will be considered invalid**.

You must use the *Least Recently Used (LRU)* page replacement strategy when allocating frames. The *LRU* implementation must use the value of the variable `t` as the clock value. The page table entry (depicted in Figure 2) contains 5 bits for *LRU* metadata storage. If the value of this field reaches 31 in any page table entry, you must reset the value of the variable `t` to 0 and set the value of the *LRU* metadata for the current page to 1. When replacing or evicting pages, you should assume that their contents have always been modified (dirty) and store them in the simulated hard drive. The page table entry must contain a validity bit at the sixth bit.

When generating random addresses, we recommend generating **even** addresses to avoid edge cases when reading from the end of a frame. You can do this by masking a randomly generated address with the value `0x3FE`.

2.1.1 Simulated CPU architecture overview

The simulated CPU uses 16-bit virtual addresses with a page/frame size of 64 bytes and the little-endian byte order. The structure of the virtual address is depicted in Figure 1. Virtual address translation is done using a single-level page table. To make the simulation easier to implement, the first 6 bits of the virtual address **are left unused**, limiting the virtual address space to **1024 bytes**. The next 4 virtual address bits are used as an index for the page table, and the last 6 bits are used as a frame offset.

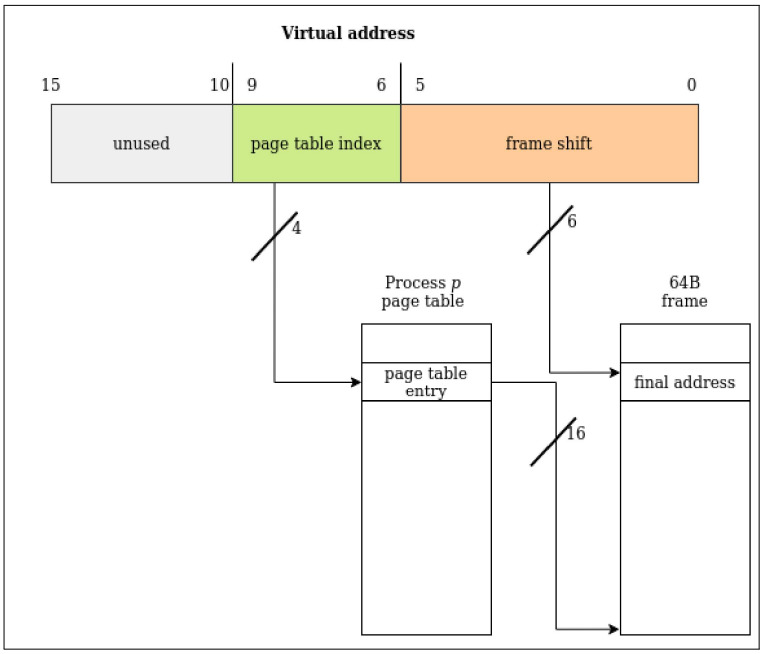

Figure 1: Simulated virtual memory system diagram.

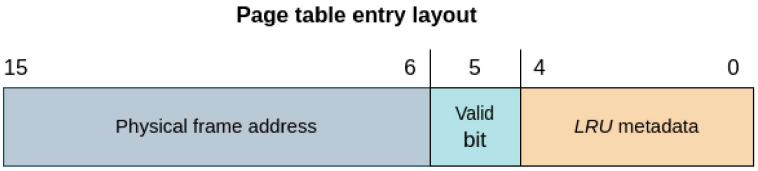

Figure 2: Page table entry layout.

2.1.2 Overview of a single simulation step

Let's assume that the randomly generated virtual address was `0x1A2`, that the corresponding page is valid, and that the value of the global clock variable **t** is 3. Finding the corresponding page table entry requires extracting several values from the virtual address. This process is depicted in Figure 3. The value of the page table index is 6, which means that we must check the seventh page table entry. The page table entry points to the frame at `0x9`, the valid bit is set and the *LRU* value is 1. The final address is then formed from the frame address and the frame shift value, and the *LRU* metadata is set to the current value of the clock (3).

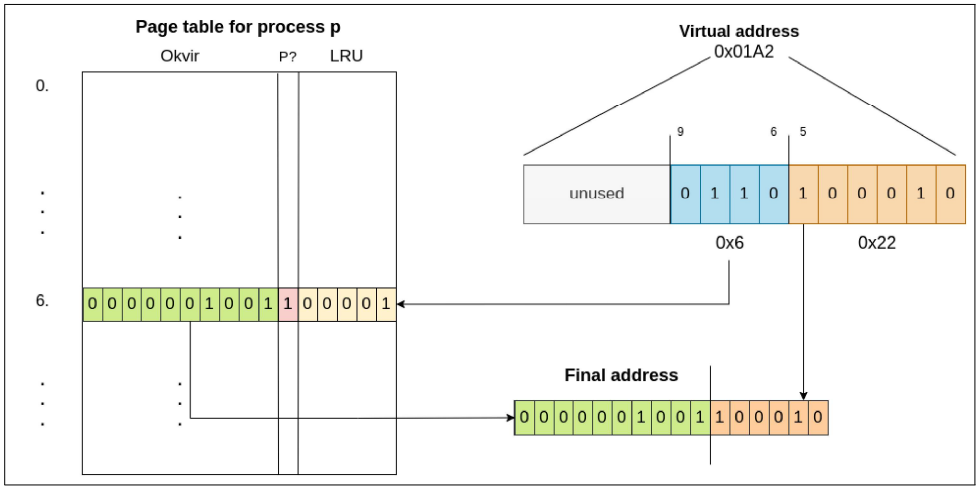

Figure 3: Address translation.

2.1.3 Sample simulation output

This is an output from a properly implemented solution simulating two processes and a single frame. Each process accessed the same address `0x1FE`.

Listing 1: Simulation output.

```
$ ./lab4 2 1
----------------------------
process: 0
        t: 0
        virt. address: 0x01fe
        Page fault!
                allocated frame 0x0000
        phys. adress: 0x003e
        page table entry: 0x0020
        address contents: 0

----------------------------
process: 1
        t: 1
        virt. address: 0x01fe
        Page fault!
                Evicting page 0x01c0 from process 0
                evicted page LRU: 0x0000
                allocated frame 0x0000
        phys. address: 0x003e
        page table entry: 0x0021
        address contents: 0

----------------------------
process: 0
        t: 2
        virt. address: 0x01fe
        Page fault!
                Evicting page 0x01c0 from process 1
    evicted page lru: 0x0001
                allocated frame 0x0000
        phys. adresa: 0x003e
        page table entry: 0x0022
        address contents: 1
----------------------------
process: 1
        t: 3
        virt. address: 0x01fe
        Page fault!
                Evicting page 0x01c0 from process 0
    evicted page lru: 0x0002
                allocated frame 0x0000
        phys. address: 0x003e
        page table entry: 0x0023
        address contents: 1
^C
```