Operating systems

Fourth laboratory exercise

March 10, 2023

Content

1 Motivation

The goal of this laboratory exercise is to study the operation of the paging system and process address space isolation in modern computers. As part of the exercise, it is necessary to simulate the operation of several processes on a simple computer that uses paging.

2 Task

2.1 Paging simulation (4 points)

Your task is to simulate the operation of multiple processes in a system that uses the on-demand paging mechanism using the architecture shown in Figure 1.

The simulated system consists of N process, disk, array of M frames and paging tables for each simulated process. Your program receives frame numbers as input parameters M and process number N and in addition to the behavior specified in pseudocode 1, it must contain the following data structures:

- disc[N] -A simulated disk used to store the content of pages,
- frame[M] -Simulated working tank of M frame size 64 octets,
- table[N] -Translation table for each of N process.

Algorithm 1: Simulation pseudocode.

```
1  for $and$=1 to $N$ does
2      create a process $and$;
3      initialize the process paging table $and$;
4  end
5  $t \leftarrow 0$;
6  repeat
7      for every process $p$ does
8          $x \leftarrow$ random logical address;
9          $and \leftarrow$ get_content($p, x$); $and \leftarrow$
10         $and+1$;
11         write_content($p, x, i$);
12         $t \leftarrow t+1$;
13         sleep;
14     end
```

```
Algorithm 2: Pseudocode of helper functions.
1 Function get_content(p, x)
2     y ← get_physical_address(p, x); and ←
3     value at address y; return and;
5
1 Function write_value(p, x, i)
2     y ← get_physical_address(p, x); write
3     down the value and to the address y;
1 Function get_physical_address(p, x)
2     find the process paging table record p for the address x;
3     if address x is not present then
4         print failure;
5         find and assign frame; load page
6         content from disk; update the process
7         translation table p; end
8
9     write the address x, the address of its frame and the content of the record in the table
        translation;
11    return physical address;
```

Auxiliary functionsfetch_contentandwrite_valueare used to access the address space of the process*p*. Any solution that accesses the simulated container in addition to these functions**will be considered invalid**.

A page replacement strategy should be used when finding a free frame to allocate*Least Recently Used (LRU)*. For implementation*LRU* strategy it is necessary to use a variablet,whose value is used as a clock. In the structure of the translation table record (shown in Figure 2), 5 bits are provided for storage*LRU*metadata for the page. If the value of that field in any page comes to 31, it is necessary to set the value of the global variabletto 0, the values of all*LRU*metadata for all records in all paging tables to 0, and*LRU*metadata for the current page to 1. When replacing or deleting pages, assume that their content is always changed and store it on disk. Each record within the paging table must achieve the presence bit in the sixth bit of the record.

For the purposes of generating a random logical address, we recommend that you generate only**pairs**addresses to avoid problematic edge cases with frame-end reads, which you can achieve by applying a boolean operation**AND** with value0x3FEto the generated number.

2.1.1 Brief description of the simulated architecture

The simulated processor uses 16-bit logical addresses with a frame/page size of 64 octets. Assume that the numbers are stored according to the principle *little-endian*. The structure of a virtual address is shown in Figure 1. For the purpose of translation of virtual addresses, a paging table with one level is used. For easier simulation, the first 6 bits of the logical address are**does not use**, which is the logical address space **limited to 1024 octets**. The next 4 bits of the virtual address are used as an index in the paging table, and the last 6 bits are used as an offset within the physical frame.

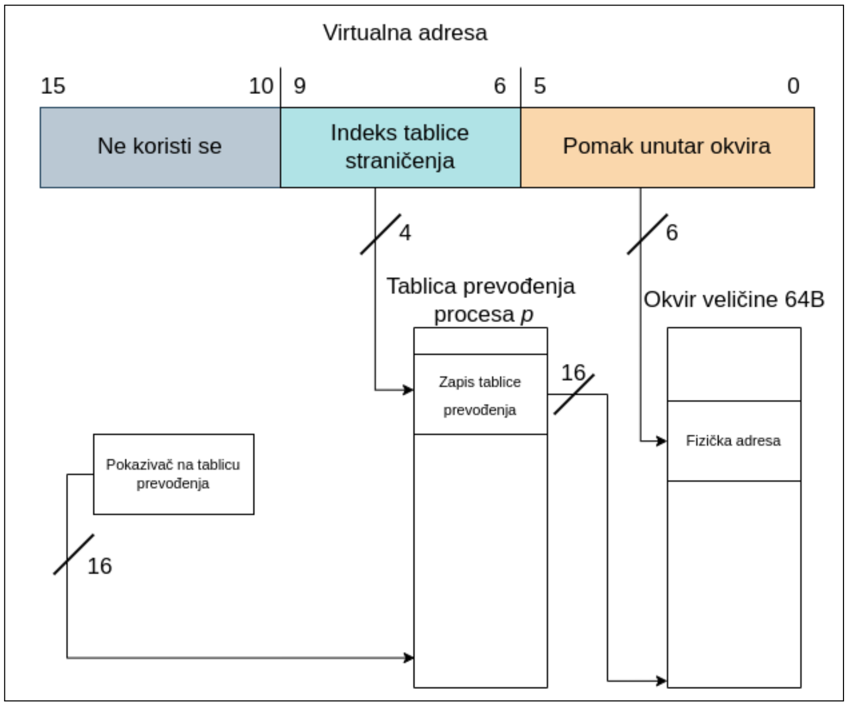

Figure 1: Paging system diagram of the simulated computer.

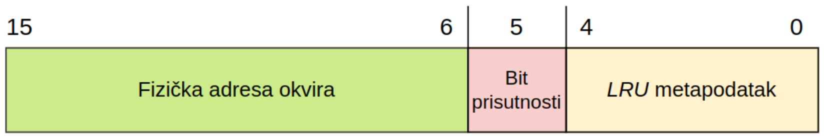

Figure 2: Structure of records in the translation table.

2.1.2 Example of one simulation step

Suppose we randomly generated an address0x1A2,that the page is present and that the value of the variable**t**=3. In order to find the corresponding record in the translation table, it is necessary to break down the address into the fields specified in Figure 1. The procedure is shown in Figure 3. The value of the obtained index is 6, and we check the seventh record in the table (counting from zero). By decomposing the value of the seventh record, we see that the address of the frame is 0x9, that the presence bit is set to 1, and that the previous*LRU*value 1. The data address is then formed from the frame address and the offset within the frame, a*LRU*data in the table record is set to 3 (the current value of the variable**t**or an hour).

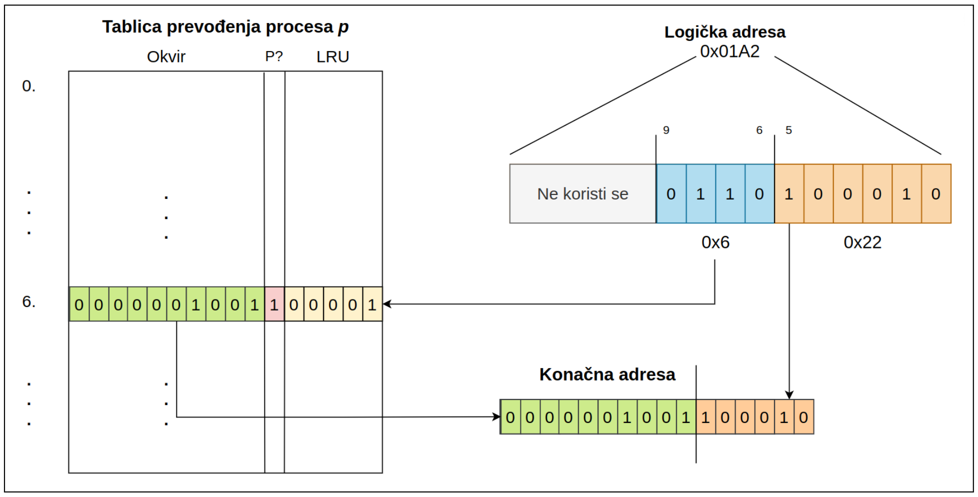

Figure 3: Example of address translation.

2.1.3 Print example

Suppose we simulate two processes with one available frame and that in four iterations of the simulation both processes accessed the address0x1FE.

Example 1: Printing the simulation.

```
$./lab4 2 1
---------------------------
process:    0
            t:0
            log. address:0x01fe Miss!

                        assigned        framework   0x0000
            Phys.   address:0x003e
            record   tables:        0x0020
            content     addresses:   0

---------------------------
process:    1
            t:1
            log. address:0x01fe Miss!

                        Ejecting page 0x01c0 from process 0 lru ejected
                        pages:0x0000 allocated
                                        frame 0x0000
            Phys.   address:0x003e
            record   tables:        0x0021
            content     addresses:   0

---------------------------
process:    0
            t:2
            log. address:0x01fe Miss!

                        Ejecting page 0x01c0 from process 1 lru ejected
                        pages:0x0001 assigned
                                        framework   0x0000
            Phys.   address:0x003e
            record   tables:        0x0022
            content     addresses:   1
---------------------------
process:    1
            t:3
            log. address:0x01fe Miss!

                        Ejecting page 0x01c0 from process 0 lru ejected
                        pages:0x0002 allocated
                                        frame 0x0000
            Phys.   address:0x003e
            record   tables:        0x0023
            content     addresses:   1
^�C
```

2.2 Simulation of a shared tank (2 points)

In the previous lab exercises, you encountered the shared container mechanism (functionsshmgetandschmat),and in this part of the exercise, your task is to study how this mechanism can be implemented using paging and to expand the simulation so that the simulated processes enable mutual communication.

2.2.1 Description of the shared tank mechanism

So far we have used the paging system so that each process has its own address space. However, such an approach does not allow processes to mutually share parts of their address space, which is necessary for the implementation of some communication mechanisms between processes. The solution to that problem is the shared container mechanism. The implementation of a shared container using paging is shown in Figure 4. A shared container is allocated a single frame that must be adequately associated with each process that wishes to use the shared container. Within each process, a free logical address is selected for the address of the container, and the frame address of the shared container is entered in the corresponding value in the paging table of that process. It is important to note here that the logical address of the container within an individual process does not have to be the same for each process,

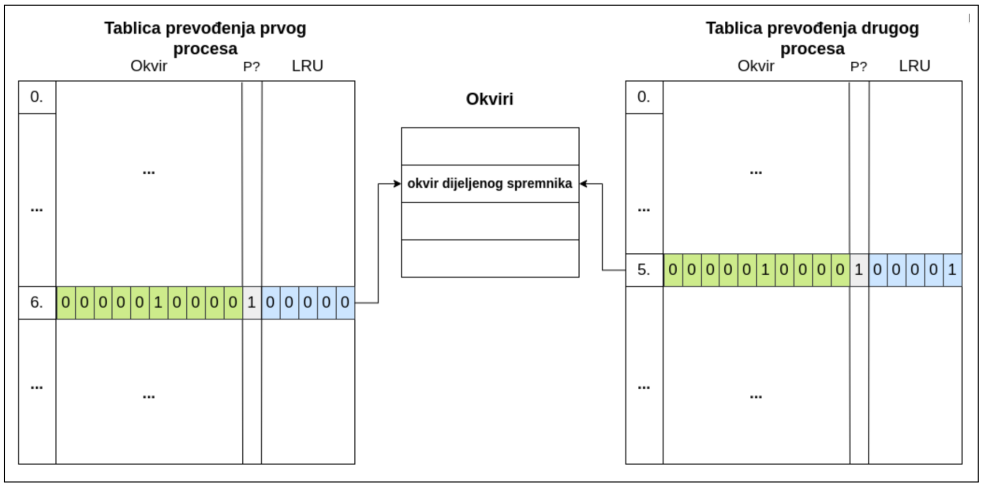
Tablica prevođenja prvog procesa
Okvir
P?
LRU
0.
...
6.
0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
Okviri
okvir dijeljenog spremnika
Tablica prevođenja drugog procesa
5.
0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1

Figure 4: Contents of the translation tables when using a shared container

2.2.2 Task

At startup, each process needs to occupy a shared container of a certain size, and your simulated system needs to populate the paging tables correctly for the shared container to work properly.

For simplicity, assume the following:

- There is always only one shared container in the entire system,
- The shared container is always one page in size, regardless of whether the entire space is used,
- Each process has the same logical address for the shared container,
- In the system, there is a separate place on the disk for saving the content of the page of the shared container,
- When selecting a frame to replace, treat the shared container frame as the lowest priority, i.e. always select it for replacement.

Using the simulated shared container mechanism, it is necessary to realize producer-consumer communication using a shared message queue, whereby the first started process is considered the producer and the others consumers. The extended simulation pseudocode is shown in Algorithm 3. The producer process sends a number of messages equal to the number of processes and places them in the common queue, while the consumer process takes messages from the common queue. This part of the task**should take place together with the simulation described in the first task**, and special attention should be paid so that random writing processes do not interfere with the operation of the shared container. As in the first part of the task, you need to access the address space of the process using functionsfetch_content andwrite_value,which means you should also use them in the part of the code that accesses the shared container. The message queue does not need to be complicated and can consist of a 2-octet message field, with each consumer process using its own identifier to access the message queue.

Pay special attention to the following cases:

- When evicting a page from a particular process, only one value within a single paging table needs to be undone. Consider what all paging table values you need to reset when you dump a shared container frame.

Algorithm 3: Extended simulation pseudocode.

```
for $and$=1 to $N$ does
    create a process $and$;
    initialize the process paging table $and$;
    occupy a common container;
end
$t$←0;
repeat
    for every process $p$ does
        if $p$ is the manufacturer then
            put $N$ −1 message to shared container;
            otherwise
            get message from shared container;
            end
        $x$←random logical address;
        $and$←get_content($p, x$); $and$←
        $and$+1;
        write_content($p, x, i$);
        $t$←$t$+1;
        sleep;
    end
```

2.2.3 Print example

Suppose we simulate two processes with one available frame and that in four iterations of the simulation both processes accessed the address0x1FE.

Example 2: Printing the extended simulation.

```
---------------------------
process:  0
          t:0
          Sent a message:4567
          Miss!
                    I dump the shared container frame allocated
                                 framework   0x0000
          Phys.  address:0x003e
          record  tables:      0x0020
          content   addresses:  0
---------------------------
process:  1
          t:1
          Miss!
                    Ejecting page 0x01c0 from process 0 lru ejected
                    pages:0x0000 allocated frame 0x0000

          Received a message:4567
          Miss!
                    I dump the shared container frame allocated
                                 framework   0x0000
          Phys.  address:0x003e
          record  tables:      0x0021
          content   addresses:  0
---------------------------
process:  0
          t:2
          Miss!
                    Ejecting page 0x01c0 from process 1 lru ejected
                    pages:0x0001 allocated frame 0x0000

          Sent a message:23c6
          Miss!
                    I dump the shared container frame allocated
                                 framework   0x0000
          Phys.  address:0x003e
          record  tables:      0x0022
          content   addresses:  1
---------------------------
process:  1
          t:3
          Miss!
                    Ejecting page 0x01c0 from process 0 lru ejected
                    pages:0x0002 allocated frame 0x0000

          Received a message:23c6
          Miss!
                    I dump the shared container frame allocated
                                 frame 0x0000
          Phys.  address:0x003e
          record  tables:      0x0023
          content   addresses:  1
^�C
```