NOVA SCHOOL OF
SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

On-demand paging

HOMEWORK 3

To be submitted for evaluation to mooshak.di.fct.unl.pt until end of 9th November 2023

This work is individual and all solutions will be compared. Don't look at other students' code and don't show your own to others.

Objective

Complete a demonstrator of a toy OS memory manager over a paging architecture. This demo only simulates the virtual addresses translation (as done by the MMU hardware) and the page-fault handling for just one process, done by an OS. We will ignore the real memory accesses, its contents, memory caches and page-table cache (TLB), etc. The main objective is to understand the handling of page faults and to simulate the on-demand page loading and page replacement policies on an OS.

Simulator Program

The provided source code already implements the address translation using a page table that must be updated at every page-fault by the OS simulator, after a new physical memory frame being allocated. The read and write of memory pages (page-in and page-out) are omitted in this simulator.

In the source code you can find the hardware paging architecture defined by the following:

PAGESIZE – architecture page size;

MAX_VIRTUAL_PAGES – the maximum page table size. That corresponds to the max address space possible of a process;

nameFrames – size of the physical memory in number of frames (physical pages);

pageTable – the process page table. The index to this array is the page number. The OS initializes it with all pages 'not present' in memory and only two 'invalid pages' (not used by the process): the page 0 and the last page of the virtual address space. These special cases are represented by special values -1 and - 2. When a page is mapped to real memory (a frame is allocated) the frame number is saved here.

The OS also maintains the following table:

frameTable – representation of the real memory status. The index to this array represents the frame number. Each position content can be -1, representing that this frame is not in use, or the number of the virtual page in this frame (note that there is only one process).

During the simulation, *simulateAllSteps* calls *translateOneAddr* to translate each virtual address to a physical address, as if done by the CPU's MMU hardware. The translation uses the *pageTable* when a frame is found for the address' page, but it can also "raise exceptions", calling the OS to handle invalid page references and page-faults.

During simulation, some statistics must be collected by incrementing some counters. Study all this code and its comments.

Work plan

Implement on-demand paging with a replacement policy. To start your implementation, consider that when a memory reference produces a page fault, the OS must verify if it's not valid or if it's just not in memory. If valid but not in memory, allocate a new empty frame for this page using a sequential policy, starting from frame 0, and update the *pageTable* and *frameTable*. At this stage, the simulator can “run” programs that fit completely in the real memory.

When all the physical memory is used (all frames are in use), a replacement policy must be used to evict a page (possibly swap it out) and use that frame for the needed page. The *pageTable* and *frameTable* must be updated as needed. Implement a page replacement solution based on the FIFO policy. In the conditions of this simulator, you can just start again, sequentially, from frame 0. Read chapter 22.3 of OSTEP book. At this stage, the simulator can “run” any program with any physical memory size.

For the final solution, change the replacement policy to give a Second Chance to recently used pages, similar do clock algorithm in OSTEP 22.8. The version to implement should be as follows:

1. at each new memory access, the MMU should mark that page as accessed (you may use a new frame table to simulate this bit);
2. when a page replacement is needed, start from a “current” position looking for an empty frame, a frame in use but without accessed bit and, if all the last choices failed, choose the “current” frame (this frame will not be marked as accessed for this first access)
3. “current” position is incremented to the next frame.

Examples and Submission

Teste in your computer. The simulator needs two arguments: first is the physical memory size given in number of memory frames; second, a text file with a sequence of memory addresses, on per line.

Some example files of memory accesses are provided and the expected output is as follows:

for mmu 2 ex1.trace:
```
swapin page 8 to frame 0
swapin page 10 to frame 1
swapout page 10 from frame 1
swapin page 18 to frame 1
swapout page 18 from frame 1
swapin page 20 to frame 1
swapout page 20 from frame 1
swapin page 28 to frame 1
Memory accesses: 9
Page faults: 5 (55.56% of memory accesses)
Swap outs: 3
```

for mmu 3 ex1.trace:
```
swapin page 8 to frame 0
swapin page 10 to frame 1
swapin page 18 to frame 2
swapout page 10 from frame 1
swapin page 20 to frame 1
swapout page 18 from frame 2
swapin page 28 to frame 2
Memory accesses: 9
Page faults: 5 (55.56% of memory accesses)
Swap outs: 2
```

for mmu 3 ex2.trace:
```
swapin page 8 to frame 0
swapin page 10 to frame 1
swapin page 18 to frame 2
swapout page 18 from frame 2
swapin page 20 to frame 2
swapout page 20 from frame 2
swapin page 18 to frame 2
```

```
swapout page 18 from frame 2
swapin page 20 to frame 2
Memory accesses: 14
Page faults: 6 (42.86% of memory accesses)
Swap outs: 3
```

for mmu 100 ex3.trace:

```
swapin page 7fff to frame 0
ERROR: invalid address: 0 (page 0)
Simulated process Segmentation fault!
Memory accesses: 3
Page faults: 1 (33.33% of memory accesses)
Swap outs: 0
```

for mmu 200 bzip.trace

```
swapin page 3d92 to frame 0
swapin page 114 to frame 1
swapin page 89 to frame 2
swapin page ff to frame 3
swapin page 116 to frame 4
swapin page 105 to frame 5
swapin page ca to frame 6
swapin page 2ff5 to frame 7
[ . . .]
swapin page 7dec to frame 5
swapout page 105 from frame 7
swapin page 7e90 to frame 7
swapout page ca from frame 8
swapin page 6f0e to frame 8
Memory accesses: 1000000
Page faults: 311 (0.03% of memory accesses)
Swap outs: 111
```

You should submit the C file with your final solution to mooshak.di.fct.unl.pt, as before. Note that all messages reporting what's happening must appear by a predefined order.

Some points will be given to a solution with page replacement using FIFO order without Second Chance.

Bibliography

OS Three Ease Pieces, chapter 22 and chapter 23 (examples of VAX/VMS and Linux)

FSO Slides