CITS2002 - Third assessed lab
A simple simulation of virtual memory

- This lab is worth 10% of the marks in the unit.
- The lab can be done in groups of two.
- The due date of the lab is **October 17, 11:59 pm**.
- The description is long, but the coding is simple. We will discuss the project in the workshops on Fridays.
- This assessed lab is extension of the second assessed lab.

1 A simple simulation of virtual memory

The aim of this lab is to simulate a simple virtual memory system using an array as the RAM of a hypothetical machine. We have a computer whose RAM is an array of size 16. It is an array of pointers. There are 16 page frames in the RAM, each consisting of one memory location in the array. Hence, the page size of this computer is 1.

The virtual memory of this computer is an array of pointers of size 32 (We will pretend it is on disc, but actually it is an array in the RAM of our computer). There are 8 processes in this computer, and each proces can have 4 pages, and obviously all the pages of all the processes cannot be in the main memory at the same time. Some pages will be in the main memory and some pages will be in the virtual memory at any time. The processes are numbered $0 \ldots 7$. Each process has a page table, which is an integer array, entry of a process page table indicates whether the page is in RAM or in the virtual memory (on disc), k if the page is in RAM (k is the frame number, between $0 \ldots 16$), and 99 if the page is in disc (99 cannot be a frame number).

You have to define a structure that will consist of three fields, a process id, a page number of the process, and the last time this page was accessed if it is in the RAM. Time in the simulation is not real time, rather a time step. Time increases in simulation steps, as explained below. The simulation starts (at time 0) by initializing the virtual memory with all the 4 pages of each process. You have to do the following steps before the simulation starts:

- Define a structure whose pointer will be stored in each array location of the RAM and the virtual memory. The structure may look like this:

```
struct {
 int process_id;
 int page_num;
 int last_accessed;
 } memory;
```

  Initialise the `process_id` and `page_num` with the id of the process (a number between $0 \ldots 7$) and a page number of that process (a number between $0 \ldots 3$). Initialise all `last_access` to 0.

- Create each page and store pointers in the array for the virtual memory.

The simulation starts by reading a file exactly as in the second assessed lab. Read the description of the second assessed lab for the meaning of this file. The only difference is that there will be only 8 processes (and 8 lines) in this file.

```
A1 90 3 7 4 2 5 1
Process2 100 3 1 4 3 6 9 7 4 9 2
```

A process requests a page whenever it is interrupted in its time quantum. The requested page is always the next page of the process. For example, if the process has no pages in the memory, the requested page is page 0. If the process has pages 1, 2, 3 in memory, the requested page is page 0, if the process has pages 1 and 2 in memory, the requested page is page 3 etc. No page is brought to the memory if all the pages of the process are already in memory at the time the process is interrupted (Note that, the page(s) of a process may be swapped out when a process is in the blocked state, but we do not care about that, we check for the requested page only when the process is blocked during a time quantum). A process always starts execution (when it is scheduled for the first time) with its page 0 in the memory.

You can keep the content of the virtual memory unchanged, as that is how virtual memory systems work. Our processes do not do any computation, they just request the next page and later may write a page back to virtual memory. You can assume for simplicity that all the pages are always in the

virtual memory and nothing needs to be written back, as no page is updated by doing any computation. The `last_accessed` time of a page will be the time step when you brought the page to RAM. For example, after reading this file, the first (or 0-th page of process 0 will be brought to RAM), the `last_accessed` time of this page will be 0, as the simulation starts now and time is 0. The `last_accessed` time for a page is the simulation time step the page is brought to memory.

The RAM may become full sometime, you have to use the local *Least Recently Used* (LRU) algorithm for evicting a page and bringing a new page. `local` means you have to evict the least recently used page of the same process for accommodating the new page. If there is no page of the process whose page you want to bring in, use a global LRU policy, evict the page that is least recently used among all pages in the RAM.

2 Submission

You have to write a C program in a single file called `simulation.c`, and compiled as an executable called `simulation`. It will read two file names from the command line, `in.txt` and `out.txt`. The first file is the one mentioned above, for reading process ids. The second file is an output file where you should print the following information at the end of the simulation. Your submission will be executed as:

```
simulation in.txt out.txt
```

- The page tables of the eight processes in separate lines, in the order from process 0 to 7. Each process will have four lines. The following examples are for process 0. The process name or number need not be printed, we will understand that from the line, for example, line 4 will be for process 3 (as process numbers start from 0). For example, the page table for process 0 may look like this:

  ```
  3, 2, 1, 99
  ```

  This means there are three pages of process 0 in the RAM, pages 0, 1 and 2, in frames 3, 2 and 1, and page 3 is in the disc. Or, like this:

  ```
  3, 99, 99, 12
  ```

This means there are two pages of process 0 in the RAM, page 0 is in frame 3 and page 3 is in frame 12.

You have to also print the content of the RAM, each location separated by a ';'. For example, the RAM may look like this:

```
0,0,5; 2,0,68; 6,3,112; etc. (16 entries)
```

Note that, the first frame of the RAM stores page 0 of process 0. Also, this page was brought to RAM at time step 5. The second frame in the RAM stores page 0 of process 2 and it was brought to RAM at time step 68; the third frame stores page 3 of process 6 and it was brought to RAM at time step 112.

Note: You have complete freedom to choose data structures that may be needed other than those mentioned in this description. But I hope you will use the code you have written for the second assessed lab.

Amitava Datta
September 2025