Indian Institute of Technology Jodhpur
Operating Systems Lab (CS330)
Assignment 7

Dated 25th April, 2021 **Total marks: 30**

Write a program in C/C++ language to implement the demand paging for virtual memory management.

The inputs to the program will be the following:
i) Total number of processes (k)
ii) Virtual address space – maximum number of pages required per process (m)
iii) Physical address space – total number of frames in maim memory (f) [m > f]
iv) Size of the TLB (s) [s < f]

Generate a random number in the range (1,k) to simulate the process dispatch during CPU scheduling. The generated number is the process ID for the current process to be executed in the CPU. The first process is dispatched without any condition. For all subsequent processes, '*the end of execution*' triggers the dispatching of the next process. For the sake of simplicity, we don't consider the implementation of CPU scheduling or context switching while handling a page fault.

For every process, your program selects a random number between (1, m) and assigns it as the required number of pages of that process and allocates frames proportionately. Per process allocation mapping can be stored in an appropriate data structure for all k processes. You can create a random mapping (adhering to the allocation policy implemented) to initialize the TLB, once the process is dispatched for the very first time.

Execution of a process means generation of page numbers from reference string. For each scheduled process P_i, generate a reference string of length lying in the range ($2*m_i$,$10*m_i$). As you parse the reference string for a specific process, for each logical page reference, first consult the TLB.

- In case of TLB hit, the corresponding frame number can be displayed in the screen with a message like "Process P_i: for page reference x, TLB hit with frame no. y".
- In case of TLB miss, consult the page table to search the entry for the referred page. If valid, the corresponding frame number can be displayed in the screen with a message like "Process P_i: for page reference x, TLB miss → page table valid→ with frame no. x".
- If the referred page is not found in the page table (invalid entry), there will be a page fault. Assume the CPU is not scheduled to serve other process while a page fault is being served.
  Invoke the page fault handler to handle the page fault
  - If free frame available - update the page table and also update the corresponding free-frame list.

- If no free frame available – do local page replacement. Select victim page using LRU, replace it and brings in a new frame and update the page table. Also, print appropriate message to reflect that page fault occurred and the update details.

You can assume printing a message in all three cases – *'TLB hit'*, *'TLB miss with no page fault'* and *'TLB miss with page fault'* is the last step for accessing a page from the reference string. Hence, the next element in the reference string can be parsed following this step.

If this is the last element in the reference string, the next process can be scheduled (generate another process ID). The completion of parsing a reference string determine 'the end of process execution'. Use signaling to communicate between the threads, if your program is multithreaded. You can also use interrupts or timers to implement these event dependencies.

Once a process is done executing, print the termination message and also print the total number of page faults for the processes.

Data Structures required:

- **TLB**
  Every time a new process is dispatched, the TLB is reloaded.
  Each entry contains <page_number, frame_number>
- **Page Table**
  For each process there is a page table
  Size of each page table is same as the size of virtual space
  Each entry in page table contains < frame_number, valid/invalid bit >
  Initially, all frame numbers are equal to -1
- **Free Frame List**
  A linked list of free frame numbers.
  Initialized to hold all frames.

Submission guidelines:

- Log the mentioned output messages at every step, in a sequential order, as your program executes and include that Result.txt file in your submission.
- Create a Readme.txt file containing all the details of *your observation, explanation, assumptions, execution requirement, limitation or bugs (if any).*
- Submit AssignmentNo._Roll.zip containing program files, readme and result file.
- **Deadline: 3rd May, 2021**

Evaluation Criteria:

- Correct implementation of the data structures:
  - TLB **(3)**
  - Page table **(3)**
  - Free-frame list **(3)**
- Correct implementation of the page allocation for all processes **(4)**

- Correct implementation of demand paging in this order TLB→Page table→Page fault handling **(6)**
- Consistency in the Result.txt **(3)**
- Proper documentation of Readme.txt **(3)**
- Overall correctness: **(5)**