README

The submitted zipped folder contains 3 files one is Lab5.cpp and Result.txt

The result in Result.txt file contains result for
k = 5 processes
m = 10 (max pages)
f = 5 (no. of frames)
s = 3 (size of TLB)

The cpp file can be run on any online IDE or IDEs like visual studio code.
For windows cmd write:
“g++ Lab5.cpp” then “a.exe”

The output will be visible on the terminal as well as in a Result.txt file (if present in the folder in which Lab5.cpp is).
Make sure to keep the Result.txt file empty before running the programme.

explanation/observation/assumptions:

- All the frames are assumed to be free in starting
- There are k processes . Which process needs to be executed at a time is chosen randomly.
- For every process a new page table is created and TLB is made randomly. Doing the mapping

  1 ---- 1
  2 ---- 2 upto s (size of TLB)

  (All the pages in TLB the are assigned the page table initially)
- The max number of pages for each process is chosen randomly between 1 to m.
- The length of ref string is chosen randomly between 2mi and 10mi
- Then the ref string is assigned pages randomly from 1 to max number of pages in process.
- Every time the ref_string is parsed , TLB is updated assuming 1 to s were the first s values in the ref string. This takes **O(size of TLB)** time.

- The TLB hit / miss function happens in **O(1)** time because unordered_map data structure is used.
- Searching the page table also takes **O(1)** time because vector<vector<int>> ds is used where index is the page no. of page table.
- When there is a page fault finding free frames takes **O(1)** time because of the linked list data structure.
- Then for page replacement the LRU algorithm is implemented which requires searching the entire page table to check which pages have frames allocated and finding the latest position of that page in ref string in backward direction. TC **in the worst case whole ref string can be searched.**
- Mostly the page replacement takes maximum time . Also we have taken LRU over here which uses past memory to find out the victim pages. Sometimes it happens if the page which was just replaced comes in the next reference string then a page replacement again needs to be performed which is a time consuming task.
- So instead of LRU , optimal page replacement algorithm which decides victim page on the basis of future req of process will decrease the the time complexity.