LEC-29: Page Replacement Algorithms

1. Whenever **Page Fault** occurs, that is, a process tries to access a page which is not currently present in a frame and OS must bring the page from swap-space to a frame.
2. OS must do page replacement to accommodate new page into a free frame, but there might be a possibility the system is working in high utilization and all the frames are busy, in that case OS must replace one of the pages allocated into some frame with the new page.
3. The **page replacement algorithm** decides which memory page is to be replaced. Some allocated page is swapped out from the frame and new page is swapped into the freed frame.
4. **Types** of Page Replacement Algorithm: (**AIM** is to have minimum page faults)
    - a. **FIFO**
        - i. Allocate frame to the page as it comes into the memory by **replacing the oldest page.**
        - ii. Easy to implement.
        - iii. Performance is **not** always good
            1. The page replaced may be an initialization module that was used long time ago **(Good replacement candidate)**
            2. The page may contain a heavily used variable that was initialized early and is in content use. (**Will again cause page fault)**
        - iv. **Belady's** anomaly is present.
            1. **In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames.** However, Balady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.
            2. This is the strange behavior shown by FIFO algorithm **in some of the cases.**
    - b. **Optimal** page replacement
        - i. Find if a page that is never referenced in future. If such a page exists, replace this page with new page.
          If no such page exists, find a page that is **referenced farthest in future**. Replace this page with new page.
        - ii. **Lowest** page fault rate among any algorithm.
        - iii. Difficult to implement as **OS requires future knowledge of reference string** which is kind of impossible. (Similar to SJF scheduling)
    - c. Least-recently used (**LRU**)
        - i. We can use recent past as an approximation of the near future then we can replace the page that has not been used for the longest period.
        - ii. Can be implemented by two ways
            1. **Counters**
                - a. Associate time field with each page table entry.
                - b. Replace the page with smallest time value.
            2. **Stack**
                - a. Keep a stack of page number.
                - b. Whenever page is referenced, it is removed from the stack & put on the top.
                - c. By this, most recently used is always on the top, & least recently used is always on the bottom.
                - d. As entries might be removed from the middle of the stack, so Doubly linked list can be used.
    - d. **Counting**-based page replacement – Keep a counter of the number of references that have been made to **each** page. (Reference counting)

i. Least frequently used (**LFU**)
   1. Actively used pages should have a large reference count.
   2. Replace page with the smallest count.

ii. Most frequently used (**MFU**)
   1. Based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

iii. Neither MFU nor LFU replacement is common.