Analysis:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **FIFO** | **LRU** | **CLOCK** |
| **cmdFile1 confFile1** | 477 | 375 | 374 |
| **cmdFile1 confFile2** | 48 | 44 | 43 |
| **cmdFile1 confFile3** | 42 | 36 | 34 |
| **cmdFile1 confFile4** | 23 | 13 | 13 |
| **cmdFile1 confFile5** | 9 | 8 | 6 |
| **cmdFile2 confFile1** | 11 | 9 | 9 |
| **cmdFile2 confFile2** | 8 | 7 | 7 |
| **cmdFile2 confFile3** | 6 | 5 | 5 |
| **cmdFile2 confFile4** | 1 | 1 | 1 |
| **cmdFile2 confFile5** | 0 | 0 | 0 |

Discussion :

7) The number of page faults of three algorithms were tested (FIFO, LRU and CLOCK) by entering the same number of pages, the same sequence and the same page frame in main memory. From above observation, and as that in; class FIFO is having the most number of page faults, whereas LRU and CLOCK result in significantly lower faults. FIFO’s logic is based on removing the oldest page in memory and while the implementation for that logic might be easy, the logic itself is rather flawed. This is because a frequently used page is often the oldest, so it will repeatedly be paged out. Throughout, all of the data tables, we saw that CLOCK results were the best results possible, and the LRU results were usually just as good, if not almost as good. This falls in line with what we’ve learnt in class that CLOCK’s performance usually falls in between LRU and FIFO. CLOCK functions like FIFO but gives each page a second chance to prove it is useful. The reason for LRU's more page faults in comparison to CLOCK can be implementation may require substantial hardware assistance. To sum up, for the given configuration we have our page fault line is ideal (kind of linear).

8) For this discussion, let’s just focus on tables in which data was collected using only the cmdFile1. It is very easy to see that with increased physical pages allocated to the process, the number of page faults decreases. Since the more frames we have in physical memory (per process) the less we have to swap pages with disc memory and thus resulting in lower overhead for the system. However, we must keep in mind that increasing the physical memory per process results in lower multiprogramming. This is where we have to be wise in using the working set strategy, where we will carefully select a resident set resulting in a lower number of page faults.