

## **Experiment No 9**

**Class:** SE Comp

**Year:** 2020-21

**Performed by:** Danyl Fernandes, 72

**Aim:** Write a program to demonstrate the concept of page replacement policies for handling page faults eg: FIFO, LRU etc.

### **Theory:**

#### **Paging:**

- Paging is a process of reading data from, and writing data to, the secondary storage.
- It is a memory management scheme that is used to retrieve processes from the secondary memory in the form of pages and store them in the primary memory. The main objective of paging is to divide each process in the form of pages of fixed size.
- These pages are stored in the main memory in frames. Pages of a process are only brought from the secondary memory to the main memory when they are needed.
- When an executing process refers to a page, it is first searched in the main memory.
- If it is not present in the main memory, a page fault occurs.
- Page Fault is the condition in which a running process refers to a page that is not loaded in the main memory.
- In such a case, the OS has to bring the page from the secondary storage into the main memory.
- This may cause some pages in the main memory to be replaced due to limited storage.
- A Page Replacement Algorithm is required to decide which page needs to be replaced.

#### **Page fault:**

- A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.
- Since actual physical memory is much smaller than virtual memory, page faults happen.
- In case of page fault, the Operating System might have to replace one of the existing pages with the newly needed page.
- Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

## **Page Replacement Policies:**

- In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.
- The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk.
- Page replacement is done when the requested page is not found in the main memory.

## **Page Replacement Algorithms:**

### **First In First Out (FIFO):**

- This is the simplest page replacement algorithm.
- In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue.
- When a page needs to be replaced, the page in the front of the queue is selected for removal.
- Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames. Find number of page faults.
- Initially all slots are empty, so when 1, 3, 0 come they are allocated to the empty slots → 3 Page Faults.
- when 3 comes, it is already in memory so → 0 Page Faults.
- Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. → 1 Page Fault.
- 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 → 1 Page Fault.
- Finally when 3 comes it is not available so it replaces 0 → 1 Page Fault
- Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm.
- For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

### **Optimal Page replacement:**

- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
- Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with a 4 page frame. Find the number of page faults.
- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots → 4 Page faults
- 0 is already there so → 0 Page fault.
- when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future. → 1 Page fault.
- 0 is already there so → 0 Page fault..
- 4 will take the place of 1 → 1 Page Fault.

- Now for the further page reference string → 0 Page fault because they are already available in the memory.
- Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests.
- The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

#### **Least Recently Used:**

- In this algorithm page will be replaced which is least recently used.
- Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames. Find number of page faults.
- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots → 4 Page faults
- 0 is already there so → 0 Page fault.
- when 3 came it will take the place of 7 because it is least recently used → 1 Page fault
- 0 is already in memory so → 0 Page fault.
- 4 will take place of 1 → 1 Page Fault
- Now for the further page reference string → 0 Page fault because they are already available in the memory.

#### **Optimality:**

- If the number of frames which are allocated to a process is not sufficient or accurate then there can be a problem of thrashing.
- Due to the lack of frames, most of the pages will be residing in the main memory and therefore more page faults will occur.
- However, if the OS allocates more frames to the process then there can be internal fragmentation.
- If the page replacement algorithm is not optimal then there will also be the problem of thrashing.
- If the number of pages that are replaced by the requested pages will be referred to in the near future then there will be more number of swap-in and swap-out and therefore the OS has to perform more replacements than usual which causes performance deficiency.
- Therefore, the task of an optimal page replacement algorithm is to choose the page which can limit the thrashing.

#### **Conclusion:**

- In this experiment, we were successfully able to demonstrate the concept of page replacement policies for handling page faults eg: FIFO, LRU etc

### **Implementation:**

```
import java.util.*;

class FifoLruImpl {

    static int fifoPageFaults(int pages[], int n, int capacity)
    {
        HashSet<Integer> s = new HashSet<>(capacity);

        Queue<Integer> indexes = new LinkedList<>();

        int page_faults = 0;
        for (int i = 0; i < n; i++) {

            if (s.size() < capacity) {

                if (!s.contains(pages[i])) {
                    s.add(pages[i]);
                    page_faults++;
                    indexes.add(pages[i]);
                }
            } else {
                if (!s.contains(pages[i])) {
                    int val = indexes.peek();

                    indexes.poll();
                    s.remove(val);
                    s.add(pages[i]);

                    indexes.add(pages[i]);
                    page_faults++;
                }
            }
        }

        return page_faults;
    }

    static int lruPageFaults(int pages[], int n, int capacity) {
        HashSet<Integer> s = new HashSet<>(capacity);
        HashMap<Integer, Integer> indexes = new HashMap<>();
```

```

int page_faults = 0;
for (int i = 0; i < n; i++) {

    if (s.size() < capacity) {
        if (!s.contains(pages[i])) {
            s.add(pages[i]);

            page_faults++;
        }

        indexes.put(pages[i], i);
    }

    else {

        if (!s.contains(pages[i])) {
            int lru = Integer.MAX_VALUE, val =
Integer.MIN_VALUE;

            Iterator<Integer> itr = s.iterator();

            while (itr.hasNext()) {
                int temp = itr.next();
                if (indexes.get(temp) < lru) {
                    lru = indexes.get(temp);
                    val = temp;
                }
            }

            s.remove(val);
            indexes.remove(val);
            s.add(pages[i]);

            page_faults++;
        }

        indexes.put(pages[i], i);
    }
}

return page_faults;

```

```

    }
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("What do you want to see
demonstrated?");
        System.out.println("1. FIFO");
        System.out.println("2. LRU");

        System.out.print("Your choice: ");
        int c = sc.nextInt();

        switch (c) {
            case 1:
                demonstrateFifo();
                break;
            case 2:
                demonstrateLru();
                break;
            default:
                System.out.println("Please enter a valid
option.");
        }
    }

    private static void demonstrateLru() {
        int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
        int capacity = 4;

        System.out.println("Page faults: " +
lruPageFaults(pages, pages.length, capacity));
    }

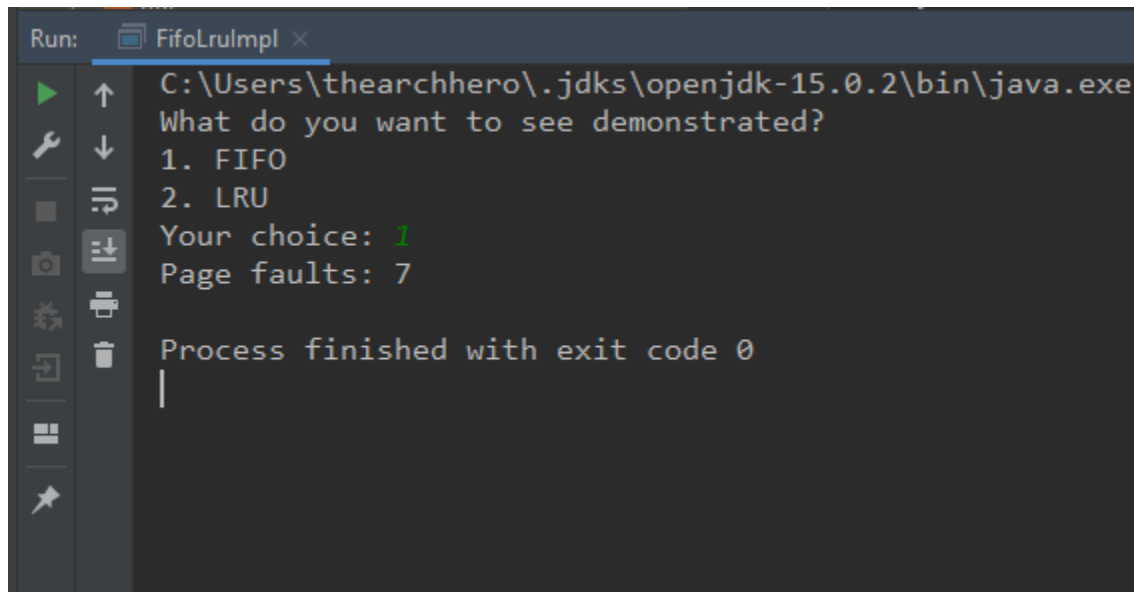
    private static void demonstrateFifo() {
        int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
        int capacity = 4;

        System.out.println("Page faults: " +
fifoPageFaults(pages, pages.length, capacity));
    }
}

```

## Output:

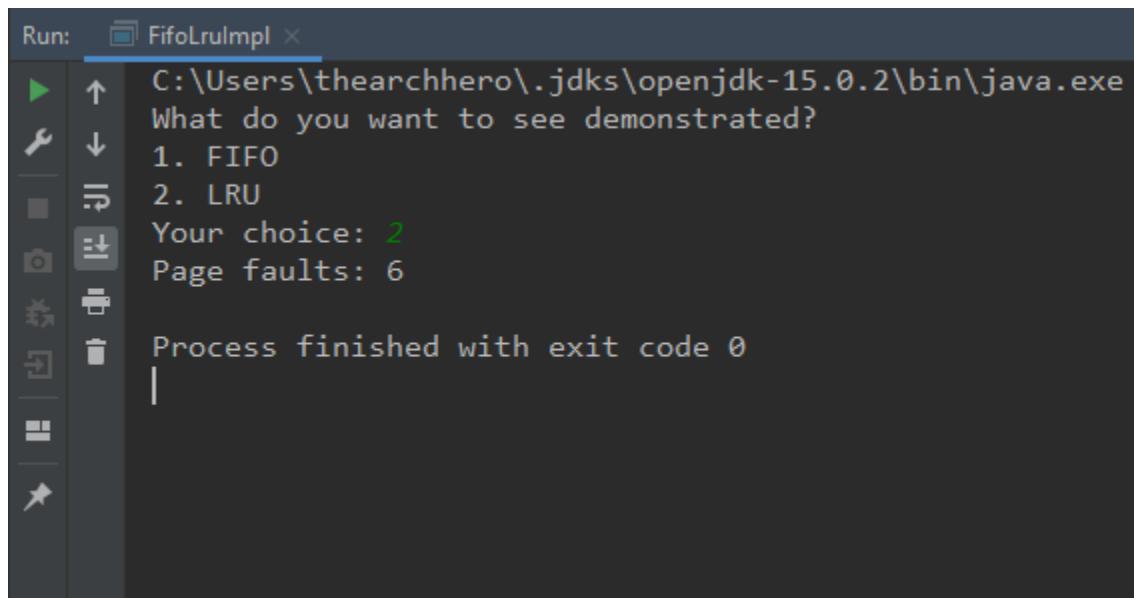
### FIFO:



```
Run: FifoLruImpl x
C:\Users\thearchhero\.jdk\openjdk-15.0.2\bin\java.exe
What do you want to see demonstrated?
1. FIFO
2. LRU
Your choice: 1
Page faults: 7

Process finished with exit code 0
```

### LRU:



```
Run: FifoLruImpl x
C:\Users\thearchhero\.jdk\openjdk-15.0.2\bin\java.exe
What do you want to see demonstrated?
1. FIFO
2. LRU
Your choice: 2
Page faults: 6

Process finished with exit code 0
```