Lab Exercise – 11

AIM :: Implement Page Replacement policy On Linux Using Shell Scripting.

Theory :: 1) FIFO Page Replacement Algorithm

- **First-In-First-Out (FIFO)** is a simple page replacement algorithm that replaces the oldest page in the memory when a new page needs to be loaded.
- The **window size** refers to the maximum number of pages that can be kept in memory at any given time.
- When a page is requested, the algorithm checks if it is already in memory. If not, a **page fault** occurs, and the oldest page is replaced.
- FIFO maintains a **queue** to track the order of page arrival, ensuring that the first page added is the first to be removed.
- FIFO is easy to implement but may suffer from the **Belady's anomaly**, where increasing the number of frames can lead to more page faults.
- It's not always optimal, as it does not consider how frequently or recently a page is accessed.

Source_Code ::

```
echo "Amit Singhal - 11614802722 (5C6)"


# Prompt user to enter the window size
echo -n "Enter the window size: "
read window_size                # Read the window size (i.e., number of frames)


# Prompt user to enter the reference string
echo -n "Enter the reference string: "
read -a ref_string              # Read reference string as an array


# Initialize empty array for frames (memory slots) and page fault counter
frames=()
page_faults=0                   # Initialize page fault counter


# Iterate through each page in the reference string
for page in "${ref_string[@]}"
do
  # Check if the page is not already in the frames (using a string comparison for array content)
```

```
    if [[ ! " ${frames[@]} " =~ " $page " ]]; then
        # If there's space in the frames (less than the window size), add the page directly
        if [ ${#frames[@]} -lt $window_size ]; then
            frames+=($page)            # Append new page to frames
        else
            # If the frames are full, remove the oldest (first) page and add the new one
            frames=("${frames[@]:1}")  # Remove the first (oldest) element from frames
            frames+=($page)            # Append new page to frames
        fi
        ((page_faults++))              # Increment the page fault count when a page replacement happens
    fi
done

# Output the number of page faults encountered
echo "Page Faults By FIFO: $page_faults"
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./amit.sh

Amit Singhal - 11614802722 (5C6)

Enter the window size: 3
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2

Page Faults By FIFO: 9

singhal-amit@singhal-amit-ThinkPad-T430:~$
```

Theory ::

2) Optimal Page Replacement Algorithm

- **Optimal Page Replacement** minimizes page faults by replacing the page that won't be needed for the longest time in the future.
- It requires **future knowledge** of memory references, which makes it theoretical and **impossible** to implement in real systems.
- When a page fault occurs, the system scans the **remaining reference string** to identify the page that will be used farthest into the future.
- Although highly efficient, this algorithm serves as a **benchmark** for evaluating other algorithms.
- It is known for delivering the **lowest number of page faults** in comparison to practical algorithms.

Source_Code ::

```
echo "Amit Singhal - 11614802722 (5C6)"

# Prompt user to enter the window size (number of frames)
echo -n "Enter the window size: "
read window_size

# Prompt user to enter the reference string (space-separated values)
echo -n "Enter the reference string (space-separated): "
read -a ref_string

# Initialize frames and page fault counter
frames=()
page_faults=0

# Iterate through each page in the reference string
for ((i=0; i<${#ref_string[@]}; i++)); do
    page=${ref_string[i]}  # Current page
    # Check if the page is already in the frames
    if [[ ! " ${frames[@]} " =~ " $page " ]]; then
        # If frames are not full, simply add the page
        if [ ${#frames[@]} -lt $window_size ]; then
            frames+=($page)
        else
            # Find the optimal page to replace
            farthest=-1
            replace_index=0
            for ((j=0; j<${#frames[@]}; j++)); do
```

```
            found=0
            for ((k=i+1; k<${#ref_string[@]}; k++)); do
                if [ ${frames[j]} -eq ${ref_string[k]} ]; then
                    if [ $k -gt $farthest ]; then
                        farthest=$k
                        replace_index=$j
                    fi
                    found=1
                    break
                fi
            done
            if [ $found -eq 0 ]; then
                replace_index=$j
                break
            fi
        done
        frames[$replace_index]=$page
    fi
    ((page_faults++))  # Increment page faults
  fi
Done

# Output the total page faults
echo "Page Faults By Optimal: $page_faults"
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./amit.sh

Amit Singhal - 11614802722 (5C6)

Enter the window size: 3
Enter the reference string (space-separated): 7 0 1 2 0 3 0 4 2 3 0 3 2

Page Faults By Optimal: 9

singhal-amit@singhal-amit-ThinkPad-T430:~$
```

Theory :: 3) Least Recently Used (LRU) Page Replacement

- **Least Recently Used (LRU)** replaces the page that has not been accessed for the longest time.
- LRU relies on the assumption that **recently used** pages will likely be used again soon.
- It tracks the **access history** of pages to identify which one was used the longest time ago.
- Though more efficient than FIFO, LRU can be harder to implement due to the need to maintain **tracking mechanisms**.
- LRU provides a good balance between performance and implementation complexity.

Source_Code ::

```
echo "Amit Singhal - 11614802722 (5C6)"

# Prompt user to enter the window size (number of frames)
echo -n "Enter the window size: "
read window_size

# Prompt user to enter the reference string (space-separated values)
echo -n "Enter the reference string (space-separated): "
read -a ref_string

# Initialize frames, usage times, and page fault counter
frames=()
usage=()
page_faults=0

# Iterate through each page in the reference string
for ((i=0; i<${#ref_string[@]}; i++)); do
    page=${ref_string[i]}  # Current page
    found=0
    # Check if the page is already in the frames
    for ((j=0; j<${#frames[@]}; j++)); do
        if [ ${frames[j]} -eq $page ]; then
            found=1
            usage[j]=$i  # Update the usage time for this page
            break
        fi
    done
```

```
    if [ $found -eq 0 ]; then
        # If frames are not full, add the page and update usage time
        if [ ${#frames[@]} -lt $window_size ]; then
            frames+=($page)
            usage+=($i)
        else
            # Find the Least Recently Used page by checking usage times
            lru_index=0
            min_usage=${usage[0]}
            for ((j=1; j<${#usage[@]}; j++)); do
                if [ ${usage[j]} -lt $min_usage ]; then
                    min_usage=${usage[j]}
                    lru_index=$j
                fi
            done
            # Replace the LRU page with the current page
            frames[$lru_index]=$page
            usage[$lru_index]=$i  # Update usage time
        fi
        ((page_faults++))  # Increment page fault counter
    fi
done

# Output the total page faults
echo "Page Faults By LRU: $page_faults"
```

Output ::

```
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./amit.sh

Amit Singhal - 11614802722 (5C6)

Enter the window size: 3
Enter the reference string (space-separated): 7 0 1 2 0 3 0 4 2 3 0 3 2

Page Faults By LRU: 8

singhal-amit@singhal-amit-ThinkPad-T430:~$
```