Ex. No. 11 a

FIFO PAGE REPLACEMENT

Aim:

To find out the number of page faults that occur using First-in First-out (FIFO) page replacement technique.

Algorithm:

1. Declare the size with respect to page length
2. Check the need of replacement from the page to memory
3. Check the need of replacement from old page to new page in memory 4.

Form a queue to hold all pages

5. Insert the page require memory into the queue
6. Check for bad replacement and page fault
7. Get the number of processes to be inserted
8. Display the values

Program Code:

```
[cse46@localhost ~]$ vi fifo_page_replacement.c
[cse46@localhost ~]$ cat fifo_page_replacement.c
#include <stdio.h>

#define MAX 50

void fifoPageReplacement(int pages[], int n, int capacity) {
    int frame[capacity];
    int pageFaults = 0;
    int index = 0;

    for (int i = 0; i < capacity; i++) {
        frame[i] = -1;
    }

    printf("Page Replacement Process:\n");

    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int isPageFound = 0;

        for (int j = 0; j < capacity; j++) {
            if (frame[j] == page) {
                isPageFound = 1;
                break;
            }
        }

        if (!isPageFound) {
            pageFaults++;

            if (frame[index] == -1) {
                frame[index] = page;
            } else {
                frame[index] = page;
            }

            printf("Frame after inserting page %d: ", page);
            for (int k = 0; k < capacity; k++) {
```

```
    printf("Page Replacement Process:\n");

    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int isPageFound = 0;

        for (int j = 0; j < capacity; j++) {
            if (frame[j] == page) {
                isPageFound = 1;
                break;
            }
        }

        if (!isPageFound) {
            pageFaults++;

            if (frame[index] == -1) {
                frame[index] = page;
            } else {
                frame[index] = page;
            }

            printf("Frame after inserting page %d: ", page);
            for (int k = 0; k < capacity; k++) {
                if (frame[k] == -1) {
                    printf(" * ");
                } else {
                    printf("%d ", frame[k]);
                }
            }
            printf("\n");

            index = (index + 1) % capacity;
        }
    }

    printf("\nTotal Page Faults: %d\n", pageFaults);
}

int main() {
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3};
    int n = sizeof(pages) / sizeof(pages[0]);
    int capacity = 3;

    fifoPageReplacement(pages, n, capacity);

    return 0;
}
```

Output:

```
[cse46@localhost ~]$ gcc fifo_page_replacement.c -o fifo_page_replacement
[cse46@localhost ~]$ ./fifo_page_replacement
Page Replacement Process:
Frame after inserting page 7: 7  *  *
Frame after inserting page 0: 7 0  *
Frame after inserting page 1: 7 0 1
Frame after inserting page 2: 2 0 1
Frame after inserting page 3: 2 3 1
Frame after inserting page 0: 2 3 0
Frame after inserting page 4: 4 3 0
Frame after inserting page 2: 4 2 0
Frame after inserting page 3: 4 2 3

Total Page Faults: 9
```

Ex. No. 11 b

LRU

Aim:

To write a c program to implement LRU page replacement algorithm.

Algorithm:

1: Start the process

2: Declare the size

3: Get the number of pages to be inserted

4: Get the value

5: Declare counter and stack

6: Select the least recently used page by counter value

7: Stack them according the selection.

8: Display the values

9: Stop the process

Program Code:

```
[cse46@localhost ~]$ vi lru_page_replacement.c
[cse46@localhost ~]$ cat lru_page_replacement.c
#include <stdio.h>

#define MAX 50

void lruPageReplacement(int pages[], int n, int capacity) {
    int frame[capacity];
    int counter[MAX];
    int pageFaults = 0;

    for (int i = 0; i < capacity; i++) {
        frame[i] = -1;
        counter[i] = 0;
    }

    printf("Page Replacement Process:\n");

    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int isPageFound = 0;
        int replaceIndex = -1;

        for (int j = 0; j < capacity; j++) {
            if (frame[j] == page) {
                isPageFound = 1;
                counter[j] = 0;
                break;
            }
        }

        if (!isPageFound) {
            pageFaults++;
            replaceIndex = 0;
            for (int j = 1; j < capacity; j++) {
                if (counter[j] > counter[replaceIndex]) {
                    replaceIndex = j;
                }
            }
            frame[replaceIndex] = page;
            counter[replaceIndex] = 0;
        }
```

```
        printf("Frame after inserting page %d: ", page);
        for (int k = 0; k < capacity; k++) {
            if (frame[k] == -1) {
                printf(" * ");
            } else {
                printf("%d ", frame[k]);
            }
        }
        printf("\n");

        for (int k = 0; k < capacity; k++) {
            if (frame[k] != -1) {
                counter[k]++;
            }
        }
    }

    printf("\nTotal Page Faults: %d\n", pageFaults);
}

int main() {
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3};
    int n = sizeof(pages) / sizeof(pages[0]);
    int capacity = 3;

    lruPageReplacement(pages, n, capacity);

    return 0;
}
```

Output:

```
[cse46@localhost ~]$ gcc lru_page_replacement.c -o lru_page_replacement
[cse46@localhost ~]$ ./lru_page_replacement
Page Replacement Process:
Frame after inserting page 7: 7  *  *
Frame after inserting page 0: 0  *  *
Frame after inserting page 1: 1  *  *
Frame after inserting page 2: 2  *  *
Frame after inserting page 0: 0  *  *
Frame after inserting page 3: 3  *  *
Frame after inserting page 0: 0  *  *
Frame after inserting page 4: 4  *  *
Frame after inserting page 2: 2  *  *
Frame after inserting page 3: 3  *  *

Total Page Faults: 10
```

Ex. No. 11 c

Optimal

Aim:

To write a c program to implement Optimal page replacement algorithm.

ALGORITHM:

1.Start the process

2.Declare the size

3.Get the number of pages to be inserted

4.Get the value

5.Declare counter and stack

6.Select the least frequently used page by counter value

7.Stack them according the selection.

8.Display the values

9.Stop the process

PROGRAM:

```
[cse46@localhost ~]$ vi optimal_page_replacement.c
[cse46@localhost ~]$ cat optimal_page_replacement.c
#include <stdio.h>

#define MAX 50

void optimalPageReplacement(int pages[], int n, int capacity) {
    int frame[capacity];
    int pageFaults = 0;

    for (int i = 0; i < capacity; i++) {
        frame[i] = -1;
    }

    printf("Page Replacement Process:\n");

    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int isPageFound = 0;
        int replaceIndex = -1;

        for (int j = 0; j < capacity; j++) {
            if (frame[j] == page) {
                isPageFound = 1;
                break;
            }
        }
```

```
        if (!isPageFound) {
            pageFaults++;
            if (i < capacity) {
                frame[i] = page;
            } else {
                int farthest = -1;

                for (int j = 0; j < capacity; j++) {
                    int k;
                    for (k = i + 1; k < n; k++) {
                        if (pages[k] == frame[j]) {
                            break;
                        }
                    }
                    if (k == n) {
                        replaceIndex = j;
                        break;
                    }
                    if (k > farthest) {
                        farthest = k;
                        replaceIndex = j;
                    }
                }

                frame[replaceIndex] = page;
            }
        }
```

```
        printf("Frame after inserting page %d: ", page);
        for (int k = 0; k < capacity; k++) {
            if (frame[k] == -1) {
                printf(" * ");
            } else {
                printf("%d ", frame[k]);
            }
        }
        printf("\n");
    }

    printf("\nTotal Page Faults: %d\n", pageFaults);
}

int main() {
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3};
    int n = sizeof(pages) / sizeof(pages[0]);
    int capacity = 3;

    optimalPageReplacement(pages, n, capacity);

    return 0;
}
```

Output:

```
[cse46@localhost ~]$ gcc optimal_page_replacement.c -o optimal_page_replacement
[cse46@localhost ~]$ ./optimal_page_replacement
Page Replacement Process:
Frame after inserting page 7: 7  *  *
Frame after inserting page 0: 7 0  *
Frame after inserting page 1: 7 0 1
Frame after inserting page 2: 2 0 1
Frame after inserting page 0: 2 0 1
Frame after inserting page 3: 2 0 3
Frame after inserting page 0: 2 0 3
Frame after inserting page 4: 2 4 3
Frame after inserting page 2: 2 4 3
Frame after inserting page 3: 2 4 3

Total Page Faults: 6
```