**Week – 12**

Implementation of following page replacement algorithms

1. FIFO 2) Optimal 3) LRU 4) LFU

**Code:**

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

void print\_slots(int slots[], int slotSize) {

for (int i = 0; i < slotSize; i++) {

if (slots[i] == -1)

printf("[ ] ");

else

printf("[%d] ", slots[i]);

}

printf("\n");

}

int FIFO(int pages[], int n, int slots[], int slotSize) {

int cur\_idx = 0;

int page\_faults = 0;

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

int page = pages[i];

bool found = false;

for (int j = 0; j < slotSize; j++) {

if (slots[j] == page) {

found = true;

break;

}

}

if (!found) {

slots[cur\_idx] = page;

cur\_idx = (cur\_idx + 1) % slotSize;

print\_slots(slots, slotSize);

page\_faults++;

}

}

return page\_faults;

}

int Optimal(int pages[], int n, int slots[], int slotSize) {

int page\_faults = 0;

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

int page = pages[i];

bool found = false;

for (int j = 0; j < slotSize; j++) {

if (slots[j] == page) {

found = true;

break;

}

}

if (!found) {

if (i < slotSize) {

slots[i] = page;

} else {

int farthest = i + 1;

int replace\_idx = -1;

for (int j = 0; j < slotSize; j++) {

int k;

for (k = i + 1; k < n; k++) {

if (pages[k] == slots[j]) {

if (k > farthest) {

farthest = k;

replace\_idx = j;

}

break;

}

}

if (k == n) {

replace\_idx = j;

break;

}

}

if (replace\_idx == -1) {

replace\_idx = 0;

}

slots[replace\_idx] = page;

}

print\_slots(slots, slotSize);

page\_faults++;

}

}

return page\_faults;

}

int LRU(int pages[], int n, int slots[], int slotSize) {

int page\_faults = 0;

int time[slotSize];

for (int i = 0; i < slotSize; i++) {

time[i] = -1;

}

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

int page = pages[i];

bool found = false;

int lru\_idx = 0;

for (int j = 0; j < slotSize; j++) {

if (slots[j] == page) {

found = true;

time[j] = i;

break;

}

if (time[j] < time[lru\_idx]) {

lru\_idx = j;

}

}

if (!found) {

slots[lru\_idx] = page;

time[lru\_idx] = i;

print\_slots(slots, slotSize);

page\_faults++;

}

}

return page\_faults;

}

int search(int r, int fa, int p[]) {

for (int i = 0; i < fa; i++) {

if (p[i] == r) {

return i;

}

}

return -1;

}

int LFU(int pages[],int n,int slots[],int slotSize){

int page\_faults = 0,insertOrder[slotSize],freq[slotSize],insertCounter = 0;

for(int i=0;i<slotSize;i++){

freq[i] = 0;

insertOrder[i] = -1;

}

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

int index = search(pages[i],slotSize,slots);

if(index != -1){

freq[index]++;

}else {

int minIndex = 0;

for (int j = 1; j < slotSize; j++) {

if (freq[j] < freq[minIndex]) {

minIndex = j;

} else if (freq[j] == freq[minIndex] && insertOrder[j] < insertOrder[minIndex]) {

minIndex = j;

}

}

slots[minIndex] = pages[i];

freq[minIndex] = 1;

insertOrder[minIndex] = insertCounter++;

page\_faults++;

print\_slots(slots,slotSize);

}

}

return page\_faults;

}

int main() {

int n, slotSize;

printf("Enter number of pages: ");

scanf("%d", &n);

int pages[n];

printf("Enter the page sequence: ");

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

scanf("%d", &pages[i]);

}

printf("Enter slot size: ");

scanf("%d", &slotSize);

int slots[slotSize];

int ch;

do {

int choice;

printf("Select 1.FIFO 2.Optimal 3.LRU 4.LFU 5.exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

for (int i = 0; i < slotSize; i++) {

slots[i] = -1;

}

int ans;

switch (choice) {

case 1:

printf("Slots:\n");

ans = FIFO(pages, n, slots, slotSize);

printf("Total no.of page faults = %d\n", ans);

break;

case 2:

printf("Slots:\n");

ans = Optimal(pages, n, slots, slotSize);

printf("Total no.of page faults = %d\n", ans);

break;

case 3:

printf("Slots:\n");

ans = LRU(pages, n, slots, slotSize);

printf("Total no.of page faults = %d\n", ans);

break;

case 4:

printf("Slots:\n");

ans = LFU(pages, n, slots, slotSize);

printf("Total no.of page faults = %d\n", ans);

break;

case 5:

printf("Done...");

return 0;

}

printf("Do you want to continue (1/0): ");

scanf("%d", &ch);

} while (ch == 1);

return 0;

}

**Output for FIFO:**

**Output for Optimal:**

**Output for LRU:**

**Output for LFU:**