**Experiment-10**

**LRU Page Replacement Algorithm**

**Code:**

#include <stdio.h>

#include <limits.h>

int checkHit(int incomingPage, int queue[], int occupied)

{

for (int i = 0; i < occupied; i++)

{

if (incomingPage == queue[i])

return 1;

}

return 0;

}

void printFrame(int queue[], int occupied)

{

for (int i = 0; i < occupied; i++)

printf("%d\t\t\t", queue[i]);

}

int main()

{

// int incomingStream[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1};

// int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3, 6, 1, 2, 4, 3};

int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};

int n = sizeof(incomingStream) / sizeof(incomingStream[0]);

int frames = 3;

int queue[n];

int distance[n];

int occupied = 0;

int pagefault = 0;

printf("Page\t Frame1 \t Frame2 \t Frame3\n");

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

{

printf("%d: \t\t", incomingStream[i]);

// what if currently in frame 7

// next item that appears also 7

// didnt write condition for HIT

if (checkHit(incomingStream[i], queue, occupied))

{

printFrame(queue, occupied);

}

// filling when frame(s) is/are empty

else if (occupied < frames)

{

queue[occupied] = incomingStream[i];

pagefault++;

occupied++;

printFrame(queue, occupied);

}

else

{

int max = INT\_MIN;

int index;

// get LRU distance for each item in frame

for (int j = 0; j < frames; j++)

{

distance[j] = 0;

// traverse in reverse direction to find

// at what distance frame item occurred last

for (int k = i - 1; k >= 0; k--)

{

++distance[j];

if (queue[j] == incomingStream[k])

break;

}

// find frame item with max distance for LRU

// also notes the index of frame item in queue

// which appears furthest(max distance)

if (distance[j] > max)

{

max = distance[j];

index = j;

}

}

queue[index] = incomingStream[i];

printFrame(queue, occupied);

pagefault++;

}

printf("\n");

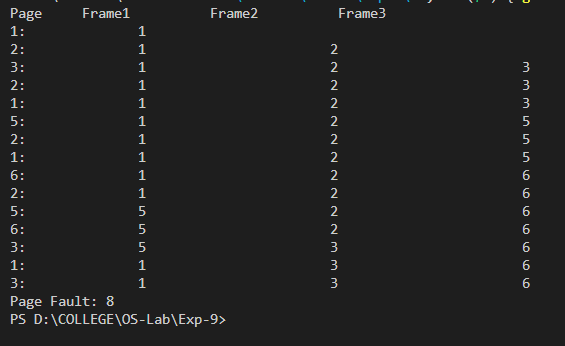
}

printf("Page Fault: %d", pagefault);

return 0;

}

**Output:**



**Optimal Page Replacement Algorithm**

**Code:**

#include <stdio.h>

// This function checks if current strea item(key) exists in any of the frames or not

int search(int key, int frame\_items[], int frame\_occupied)

{

for (int i = 0; i < frame\_occupied; i++)

if (frame\_items[i] == key)

return 1;

return 0;

}

void printOuterStructure(int max\_frames)

{

printf("Stream ");

for (int i = 0; i < max\_frames; i++)

printf("Frame%d ", i + 1);

}

void printCurrFrames(int item, int frame\_items[], int frame\_occupied, int max\_frames)

{

// print current reference stream item

printf("\n%d \t\t", item);

// print frame occupants one by one

for (int i = 0; i < max\_frames; i++)

{

if (i < frame\_occupied)

printf("%d \t\t", frame\_items[i]);

else

printf("- \t\t");

}

}

// This Function helps in finding frame that will not be used

// for the longest period of time in future in ref\_str[0 ... refStrLen - 1]

int predict(int ref\_str[], int frame\_items[], int refStrLen, int index, int frame\_occupied)

{

// For each current occupant in frame item

// we try to find the frame item that will not be referenced in

// for the longest in future in the upcoming reference string

int result = -1, farthest = index;

for (int i = 0; i < frame\_occupied; i++)

{

int j;

for (j = index; j < refStrLen; j++)

{

if (frame\_items[i] == ref\_str[j])

{

if (j > farthest)

{

farthest = j;

result = i;

}

break;

}

}

// If we find a page that is never referenced in future,

// return it immediately as its the best

if (j == refStrLen)

return i;

}

// If none of the frame items appear in reference string

// in the future then we return 0th index. Otherwise we return result

return (result == -1) ? 0 : result;

}

void optimalPage(int ref\_str[], int refStrLen, int frame\_items[], int max\_frames)

{

// initially none of the frames are occupied

int frame\_occupied = 0;

printOuterStructure(max\_frames);

// Here we traverse through reference string

// and check for miss and hit.

int hits = 0;

for (int i = 0; i < refStrLen; i++)

{

// If found already in the frame items : HIT

if (search(ref\_str[i], frame\_items, frame\_occupied))

{

hits++;

printCurrFrames(ref\_str[i], frame\_items, frame\_occupied, max\_frames);

continue;

}

// If not found in frame items : MISS

// If frames are empty then current reference string item in frame

if (frame\_occupied < max\_frames)

{

frame\_items[frame\_occupied] = ref\_str[i];

frame\_occupied++;

printCurrFrames(ref\_str[i], frame\_items, frame\_occupied, max\_frames);

}

// else we need to use optmial algorithm to find

// frame index where we need to do replacement for this

// incoming reference string item

else

{

int pos = predict(ref\_str, frame\_items, refStrLen, i + 1, frame\_occupied);

frame\_items[pos] = ref\_str[i];

printCurrFrames(ref\_str[i], frame\_items, frame\_occupied, max\_frames);

}

}

printf("\n\nHits: %d\n", hits);

printf("Misses: %d", refStrLen - hits);

}

// Driver Function

int main()

{

// int ref\_str[] = {9, 0, 5, 1, 0, 3, 0, 4, 1, 3, 0, 3, 1, 3};

int ref\_str[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1};

int refStrLen = sizeof(ref\_str) / sizeof(ref\_str[0]);

int max\_frames = 3;

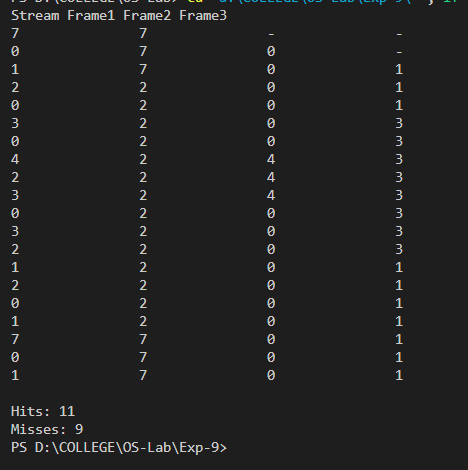
int frame\_items[max\_frames];

optimalPage(ref\_str, refStrLen, frame\_items, max\_frames);

return 0;

}

**Output:**



**LFU Page Replacement Algorithm**

**Code:**

#include <stdio.h>

void print(int frameno, int frame[])

{

int j;

for (j = 0; j < frameno; j++)

printf("%d\t", frame[j]);

printf("\n");

}

int main()

{

int i, j, k, n, page[50], frameno, frame[10], move = 0, flag, count = 0, count1[10] = {0},

repindex, leastcount;

float rate;

printf("Enter the number of pages\n");

scanf("%d", &n);

printf("Enter the page reference numbers\n");

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

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

printf("Enter the number of frames\n");

scanf("%d", &frameno);

for (i = 0; i < frameno; i++)

frame[i] = -1;

printf("Page reference string\tFrames\n");

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

{

printf("%d\t\t\t", page[i]);

flag = 0;

for (j = 0; j < frameno; j++)

{

if (page[i] == frame[j])

{

flag = 1;

count1[j]++;

printf("No replacement\n");

break;

}

}

if (flag == 0 && count < frameno)

{

frame[move] = page[i];

count1[move] = 1;

move = (move + 1) % frameno;

count++;

print(frameno, frame);

}

else if (flag == 0)

{

repindex = 0;

leastcount = count1[0];

for (j = 1; j < frameno; j++)

{

if (count1[j] < leastcount)

{

repindex = j;

leastcount = count1[j];

}

}

frame[repindex] = page[i];

count1[repindex] = 1;

count++;

print(frameno, frame);

}

}

rate = (float)count / (float)n;

printf("Number of page faults is %d\n", count);

printf("Fault rate is %f\n", rate);

return 0;

}

**Output:**

