Operating systems lab

Ex 10 – page replacement algorithms

Sreyas v

Cse c 185001162

MAIN PROGRAM

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include "LinkedList.h"

#define ROW 10

#define COL 20

int \*const convert(const char \*const, int \*);

void FIFO(const int \*const, const int, const int);

void optimal(const int \*const, const int, const int);

void LRU(const int \*const, const int, const int);

void LFU(const int \*const, const int, const int);

void putTable(const int[ROW][COL], const int, const int);

int main()

{

int n\_free\_frames = -1;

int n\_reqd\_frames = -1;

char buffer[20] = {0};

int \*sequence = NULL;

int choice = -1;

int len = 0;

while (1)

{

printf("\t\t\t\tPAGE REPLACEMENT TECHNIQUES\n");

printf(" 1 - Read Input\n");

printf(" 2 - FIFO\n");

printf(" 3 - Optimal\n");

printf(" 4 - LRU\n");

printf(" 5 - LFU\n");

printf(" 0 - Exit\n");

printf(" -------------------------\n");

printf(" Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 0:

exit(0);

case 1:

printf(" Enter the number of free frames: ");

scanf("%d", &n\_free\_frames);

printf(" Enter the number of required frames: ");

scanf("%d", &n\_reqd\_frames);

getchar();

printf(" Enter the Reference String: ");

scanf("%[^\n]", buffer);

sequence = convert(buffer, &len);

break;

case 2:

printf("\n\t\tFIFO\n");

FIFO(sequence, len, n\_reqd\_frames);

break;

case 3:

printf("\n\t\t\tOPTIMAL\n");

optimal(sequence, len, n\_reqd\_frames);

break;

case 4:

printf("\n\t\tLRU\n");

LRU(sequence, len, n\_reqd\_frames);

break;

case 5:

printf("\n\t\tLFU\n");

LFU(sequence, len, n\_reqd\_frames);

break;

default:

printf(" Invalid Input!\n");

}

printf("\n");

}

}

int \*const convert(const char \*const refstr, int \*size)

{

static int arr[30];

int i = 0, val = 0;

while (refstr[i])

{

if (isdigit(refstr[i]))

{

val = refstr[i] - 48;

for (int j = i + 1; refstr[j] && isdigit(refstr[j]); j++)

{

val = (val \* 10) + (refstr[j] - 48);

i = j;

}

arr[\*size] = val;

(\*size)++;

}

i++;

}

return arr;

}

void putTable(const int table[ROW][COL], const int n\_frames, const int n\_updates)

{

printf("\n ");

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

printf("+----");

printf("+\n ");

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

{

for (int j = 0; j < n\_updates; j++)

{

if (table[i][j] == -1)

printf("| - ");

else

printf("| %-2d ", table[i][j]);

}

printf("|\n ");

}

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

printf("+----");

printf("+\n ");

}

void insertTable(List tmp, int table[ROW][COL], const int n\_frames, const int faults)

{

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

{

if (tmp)

{

table[i][faults] = tmp->d;

tmp = tmp->next;

}

else

table[i][faults] = -1;

}

}

void FIFO(const int \*const seq, const int len, const int n\_frames)

{

int size = 0;

int faults = 0;

int table[ROW][COL];

List alloc = createEmptyList();

Node \*oldest;

printf("\n");

printf(" Frame -> In Memory -> Faults \n\n");

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

{

printf(" %-2d ->", seq[i]);

Node \*isFound = search(alloc, seq[i]);

Node \*tmp;

if (!isFound)

{

if (size < n\_frames)

{

insertLast(alloc, seq[i]);

size++;

//Initialise first frame as oldest

if (size == 1)

oldest = alloc->next;

}

else

{

//Swap oldest frame with new frame

oldest->d = seq[i];

//Update oldest frame

if (oldest->next)

oldest = oldest->next;

else

oldest = alloc->next;

}

//Updating Table

insertTable(alloc -> next, table, n\_frames, faults);

faults++;

}

display(alloc);

for (int i = length(alloc) \* 3; i <= 22; i++)

printf(" ");

printf("-> %-2d \n", faults);

}

putTable(table, n\_frames, faults);

}

void optimal(const int \*const seq, const int len, const int n\_frames)

{

int size = 0;

int faults = 0;

int distance;

int flag;

int table[ROW][COL];

List alloc = createEmptyList();

Node \*farthest = NULL, \*tmp;

printf("\n");

printf(" Frame -> In Memory -> Faults \n\n");

int val = 0;

int i = 0;

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

{

printf(" %-2d ->", seq[i]);

Node \*isFound = search(alloc, seq[i]);

if (!isFound)

{

if (size < n\_frames)

{

insertLast(alloc, seq[i]);

size++;

}

else

{

tmp = alloc->next;

distance = 0;

//Find the frame which is used the farthest away and swap

while (tmp)

{

flag = 0;

for (int j = i + 1; j < len; j++)

{

if (seq[j] == tmp->d)

{

flag = 1;

if (j - i > distance)

{

distance = (j - i);

farthest = tmp;

}

break;

}

}

//Not Used in the future

if (!flag)

{

farthest = tmp;

break;

}

tmp = tmp->next;

}

farthest->d = seq[i];

}

//Updating Table

insertTable(alloc -> next, table, n\_frames, faults);

faults++;

}

display(alloc);

for (int i = length(alloc) \* 3; i <= 22; i++)

printf(" ");

printf("-> %-2d \n", faults);

}

putTable(table, n\_frames, faults);

}

void LRU(const int \*const seq, const int len, const int n\_frames)

{

int size = 0;

int faults = 0;

int distance;

int table[ROW][COL];

List alloc = createEmptyList();

Node \*least\_recent = NULL, \*tmp;

printf("\n");

printf(" Frame -> In Memory -> Faults \n\n");

int val = 0;

int i = 0;

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

{

printf(" %-2d ->", seq[i]);

Node \*isFound = search(alloc, seq[i]);

if (!isFound)

{

if (size < n\_frames)

{

insertLast(alloc, seq[i]);

size++;

}

else

{

tmp = alloc->next;

distance = 0;

//Find the frame which is used the least recently and swap

while (tmp)

{

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

{

if (seq[j] == tmp->d)

{

if (i - j > distance)

{

distance = (i - j);

least\_recent = tmp;

}

break;

}

}

tmp = tmp->next;

}

least\_recent->d = seq[i];

}

//Updating Table

insertTable(alloc -> next, table, n\_frames, faults);

faults++;

}

display(alloc);

for (int i = length(alloc) \* 3; i <= 22; i++)

printf(" ");

printf("-> %-2d \n", faults);

}

putTable(table, n\_frames, faults);

}

void LFU(const int \*const seq, const int len, const int n\_frames)

{

int size = 0;

int faults = 0;

int frequency;

int table[ROW][COL];

List alloc = createEmptyList();

Node \*least\_frequent = NULL, \*tmp;

printf("\n");

printf(" Frame -> In Memory -> Faults \n\n");

int val = 0;

int i = 0;

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

{

printf(" %-2d ->", seq[i]);

Node \*isFound = search(alloc, seq[i]);

if (!isFound)

{

if (size < n\_frames)

{

insertLast(alloc, seq[i]);

size++;

}

else

{

tmp = alloc->next;

frequency = 99;

//Find the frame which is least frequently used and swap

while (tmp)

{

if (tmp->freq < frequency)

{

frequency = tmp->freq;

least\_frequent = tmp;

}

tmp = tmp->next;

}

least\_frequent->d = seq[i];

least\_frequent->freq = 1;

}

//Updating Table

insertTable(alloc -> next, table, n\_frames, faults);

faults++;

}

else

isFound->next->freq++;

display(alloc);

for (int i = length(alloc) \* 3; i <= 22; i++)

printf(" ");

printf("-> %-2d \n", faults);

}

putTable(table, n\_frames, faults);

}

OUTPUT

Main Menu

1. Enter the Reference String

2. View the Reference String

3. Implement FIFO Algorithm

4. Implement LRU Algorithm

5. Implement Optimal Algorithm

6. Implement LFU Algorithm

0. Exit

Your Choice -> 1

Enter the length of the reference string(Maximum : 20): 20

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

Enter the frame size: 3

Main Menu

1. Enter the Reference String

2. View the Reference String

3. Implement FIFO Algorithm

4. Implement LRU Algorithm

5. Implement Optimal Algorithm

6. Implement LFU Algorithm

0. Exit

Your Choice -> 3

Implementing FIFO Algorithm:

7

0 7

1 0 7

2 1 0

2 1 0

3 2 1

0 3 2

4 0 3

2 4 0

3 2 4

0 3 2

0 3 2

0 3 2

1 0 3

2 1 0

2 1 0

2 1 0

7 2 1

0 7 2

1 0 7

No. of Page Faults on performing FIFO Algorithm: 15

Main Menu

1. Enter the Reference String

2. View the Reference String

3. Implement FIFO Algorithm

4. Implement LRU Algorithm

5. Implement Optimal Algorithm

6. Implement LFU Algorithm

0. Exit

Your Choice -> 4

Implementing Least Recently Used Algorithm:

7

0 7

1 0 7

2 1 0

2 1 0

3 2 0

3 2 0

4 3 0

2 4 0

3 2 4

0 3 2

0 3 2

0 3 2

1 3 2

1 3 2

0 1 2

0 1 2

7 0 1

7 0 1

7 0 1

No. of Page Faults on performing LRU Algorithm: 12

Main Menu

1. Enter the Reference String

2. View the Reference String

3. Implement FIFO Algorithm

4. Implement LRU Algorithm

5. Implement Optimal Algorithm

6. Implement LFU Algorithm

0. Exit

Your Choice -> 5

Performing Optimal Algorithm:

7

0 7

1 0 7

2 0 7

2 0 7

3 2 7

0 3 2

4 3 2

4 3 2

4 3 2

0 3 2

0 3 2

0 3 2

1 0 2

1 0 2

1 0 2

1 0 2

7 1 0

7 1 0

7 1 0

No. of Page Faults on performing Optimal Algorithm: 10

Main Menu

1. Enter the Reference String

2. View the Reference String

3. Implement FIFO Algorithm

4. Implement LRU Algorithm

5. Implement Optimal Algorithm

6. Implement LFU Algorithm

0. Exit

Your Choice -> 6

Implementing Least Frequently Used Algorithm:

7

0 7

1 0 7

2 1 0

2 1 0

3 2 0

3 2 0

4 3 0

2 4 0

3 2 0

3 2 0

3 2 0

3 2 0

1 3 0

2 3 0

2 3 0

1 2 0

7 2 0

7 2 0

1 2 0

No. of Page Faults on performing LFU Algorithm: 13

Main Menu

1. Enter the Reference String

2. View the Reference String

3. Implement FIFO Algorithm

4. Implement LRU Algorithm

5. Implement Optimal Algorithm

6. Implement LFU Algorithm

0. Exit

Your Choice -> 0

Thank You!