**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**Semester - IV**

**UCS1411 – Operating Systems Lab**

**Academic Year: 2019-2020 Batch: 2018-2022**

**Name : Rahul Ram M**

**Class : CSE – B**

**Register Number : 185001121**

**Lab Exercise 10 Implementation of Page Replacement Algorithms**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

struct Node

{

int data;

struct Node \*next;

};

int findLFU(struct Node\* node, int arr[], int start, int len){

int min = 10000;

int value;

int count;

while(node != NULL){

count = 0;

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

if(node->data == arr[i]){

count++;

}

}

if(count < min){

min = count;

value = node->data;

}

node = node->next;

}

return value;

}

int findLRU(struct Node\* node, int arr[], int start, int len){

int min = 10000;

int value;

int count;

while(node != NULL){

count = 0;

for(int i = start; i >= 0; i--){

if(node->data == arr[i]){

break;

}

count++;

}

if(start - count < min){

min = start-count;

value = node->data;

}

node = node->next;

}

return value;

}

int findOptimal(struct Node\* node, int arr[], int start, int len){

int max = -1;

int count;

int value;

while(node != NULL){

count = 0;

for(int i = start; i < len; i++){

if(node->data == arr[i]){

break;

}

count++;

}

if(count > max){

max = count;

value = node->data;

}

node = node->next;

}

return value;

}

void printString(int arr[], int len){

printf("Reference String : ");

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

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

}

printf("\n\n");

}

void printTable(int\*\* arr, int len, int width){

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

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

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

printf(" ");

else

printf(" %d ",arr[j][i]);

}

printf("\n");

}

}

bool checkList(struct Node\* node, int value){

while(node != NULL){

if(node->data == value)

return true;

node = node->next;

}

return false;

}

void copyList(struct Node\* node, int\*\* arr, int index, int len){

for(int i = 0; i < len && node != NULL; i++){

arr[index][i] = node->data;

node = node->next;

}

}

void append(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

struct Node \*last = \*head\_ref;

new\_node->data = new\_data;

new\_node->next = NULL;

if (\*head\_ref == NULL) {

\*head\_ref = new\_node;

return;

}

while (last->next != NULL)

last = last->next;

last->next = new\_node;

return;

}

void replaceFrame(struct Node \*\*head\_ref, int key, int page){

struct Node\* temp = \*head\_ref;

while(temp != NULL){

if(temp->data == key){

temp->data = page;

return;

}

temp = temp->next;

}

}

void printList(struct Node \*node) {

while (node != NULL) {

printf(" %d ", node->data);

node = node->next;

}

}

int main(){

int frames;

int required;

int pages[50];

int numpages;

int choice = 1;

while(choice != 6){

printf("\n1.READ INPUT\n2.FIFO\n3.OPTIMAL\n4.LRU\n5.LFU\n6.Exit\nEnter your choice : ");

scanf("%d",&choice);

switch(choice){

case 1:{

char refstr[50], ch;

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

scanf("%d",&frames);

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

scanf("%d",&required);

if(frames < required){

printf("Frames required by proecess is greater than the free frames!\n");

break;

}

scanf("%c",&ch);

printf("Enter the reference string: ");

scanf("%[^\n]%\*c", refstr);

char\* token = strtok(refstr, " ");

numpages = 0;

while(token != NULL){

pages[numpages++] = atoi(token);

token = strtok(NULL, " ");

}

break;

}

case 2:{

printf("\t\tFIFO Page Replacement Algorithm\n\n");

printString(pages, numpages);

struct Node\* fifo = NULL;

int i;

int last[required];

int count = 0;

int \*\*table = (int \*\*)malloc(numpages \* sizeof(int \*));

for (int r = 0; r < numpages; r++)

table[r] = (int \*)malloc(required \* sizeof(int));

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

for(int k = 0; k < required; k++){

table[j][k] = -1;

}

}

int page\_faults = 0;

printf("Page ref-> memory ->PF\n");

for(i = 0; i < required; i++){

page\_faults++;

append(&fifo, pages[i]);

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

printList(fifo);

copyList(fifo, table, i, required);

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

printf(" - ");

}

printf(" -> %d\n",page\_faults);

last[i] = pages[i];

}

count = required;

for(i = i; i < numpages; i++){

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

if(!checkList(fifo, pages[i])){

replaceFrame(&fifo, last[0], pages[i]);

copyList(fifo, table, count++, required);

printList(fifo);

page\_faults++;

printf(" -> %d\n",page\_faults);

for(int k = 0; k < required-1; k++)

last[k] = last[k+1];

last[required-1] = pages[i];

}

else{

printList(fifo);

printf(" -> -\n");

}

}

printf("\n");

printTable(table, count, required);

printf("\n");

printf("Total number of page faults : %d\n",page\_faults);

break;

}

case 3:{

printf("\t\tOptimal Page Replacement Algorithm\n\n");

printString(pages, numpages);

struct Node\* optimal = NULL;

int i;

int count = 0;

int \*\*table = (int \*\*)malloc(numpages \* sizeof(int \*));

for (int r = 0; r < numpages; r++)

table[r] = (int \*)malloc(required \* sizeof(int));

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

for(int k = 0; k < required; k++){

table[j][k] = -1;

}

}

int page\_faults = 0;

printf("Page ref-> memory ->PF\n");

for(i = 0; i < required; i++){

page\_faults++;

append(&optimal, pages[i]);

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

printList(optimal);

copyList(optimal, table, i, required);

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

printf(" - ");

}

printf(" -> %d\n",page\_faults);

}

count = required;

for(i = i; i < numpages; i++){

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

if(!checkList(optimal, pages[i])){

int val = findOptimal(optimal, pages, i, numpages);

replaceFrame(&optimal, val, pages[i]);

copyList(optimal, table, count++, required);

printList(optimal);

page\_faults++;

printf(" -> %d\n",page\_faults);

}

else{

printList(optimal);

printf(" -> -\n");

}

}

printf("\n");

printTable(table, count, required);

printf("\n");

printf("Total number of page faults : %d\n",page\_faults);

break;

}

case 4:{

printf("\t\tLRU Page Replacement Algorithm\n\n");

printString(pages, numpages);

struct Node\* lru = NULL;

int i;

int count = 0;

int \*\*table = (int \*\*)malloc(numpages \* sizeof(int \*));

for (int r = 0; r < numpages; r++)

table[r] = (int \*)malloc(required \* sizeof(int));

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

for(int k = 0; k < required; k++){

table[j][k] = -1;

}

}

int page\_faults = 0;

printf("Page ref-> memory ->PF\n");

for(i = 0; i < required; i++){

page\_faults++;

append(&lru, pages[i]);

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

printList(lru);

copyList(lru, table, i, required);

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

printf(" - ");

}

printf(" -> %d\n",page\_faults);

}

count = required;

for(i = i; i < numpages; i++){

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

if(!checkList(lru, pages[i])){

int val = findLRU(lru, pages, i, numpages);

replaceFrame(&lru, val, pages[i]);

copyList(lru, table, count++, required);

printList(lru);

page\_faults++;

printf(" -> %d\n",page\_faults);

}

else{

printList(lru);

printf(" -> -\n");

}

}

printf("\n");

printTable(table, count, required);

printf("\n");

printf("Total number of page faults : %d\n",page\_faults);

break;

}

case 5:{

printf("\t\tLFU Page Replacement Algorithm\n\n");

printString(pages, numpages);

struct Node\* lfu = NULL;

int i;

int count = 0;

int \*\*table = (int \*\*)malloc(numpages \* sizeof(int \*));

for (int r = 0; r < numpages; r++)

table[r] = (int \*)malloc(required \* sizeof(int));

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

for(int k = 0; k < required; k++){

table[j][k] = -1;

}

}

int page\_faults = 0;

printf("Page ref -> memory -> PF\n");

for(i = 0; i < required; i++){

page\_faults++;

append(&lfu, pages[i]);

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

printList(lfu);

copyList(lfu, table, i, required);

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

printf(" - ");

}

printf(" -> %d\n",page\_faults);

}

count = required;

for(i = i; i < numpages; i++){

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

if(!checkList(lfu, pages[i])){

int val = findLFU(lfu, pages, i, numpages);

replaceFrame(&lfu, val, pages[i]);

copyList(lfu, table, count++, required);

printList(lfu);

page\_faults++;

printf(" -> %d\n",page\_faults);

}

else{

printList(lfu);

printf(" -> -\n");

}

}

printf("\n");

printTable(table, count, required);

printf("\n");

printf("Total number of page faults : %d\n",page\_faults);

break;

}

case 6:{

printf("Program terminated Successfully!\n");

break;

}

default:printf("Invalid Input!\n");

}

}

return 0;

}

**Sample I/O:**

1.READ INPUT

2.FIFO

3.OPTIMAL

4.LRU

5.LFU

6.Exit

Enter your choice : 1

Enter the number of free frames: 10

Enter the number of frames required by the process: 3

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

1.READ INPUT

2.FIFO

3.OPTIMAL

4.LRU

5.LFU

6.Exit

Enter your choice : 2

FIFO Page Replacement Algorithm

Reference String : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page ref-> memory ->PF

7 -> 7 - - -> 1

0 -> 7 0 - -> 2

1 -> 7 0 1 -> 3

2 -> 2 0 1 -> 4

0 -> 2 0 1 -> -

3 -> 2 3 1 -> 5

0 -> 2 3 0 -> 6

4 -> 4 3 0 -> 7

2 -> 4 2 0 -> 8

3 -> 4 2 3 -> 9

0 -> 0 2 3 -> 10

3 -> 0 2 3 -> -

2 -> 0 2 3 -> -

1 -> 0 1 3 -> 11

2 -> 0 1 2 -> 12

0 -> 0 1 2 -> -

1 -> 0 1 2 -> -

7 -> 7 1 2 -> 13

0 -> 7 0 2 -> 14

1 -> 7 0 1 -> 15

7 7 7 2 2 2 4 4 4 0 0 0 7 7 7

0 0 0 3 3 3 2 2 2 1 1 1 0 0

1 1 1 0 0 0 3 3 3 2 2 2 1

Total number of page faults : 15

1.READ INPUT

2.FIFO

3.OPTIMAL

4.LRU

5.LFU

6.Exit

Enter your choice : 3

Optimal Page Replacement Algorithm

Reference String : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page ref-> memory ->PF

7 -> 7 - - -> 1

0 -> 7 0 - -> 2

1 -> 7 0 1 -> 3

2 -> 2 0 1 -> 4

0 -> 2 0 1 -> -

3 -> 2 0 3 -> 5

0 -> 2 0 3 -> -

4 -> 2 4 3 -> 6

2 -> 2 4 3 -> -

3 -> 2 4 3 -> -

0 -> 2 0 3 -> 7

3 -> 2 0 3 -> -

2 -> 2 0 3 -> -

1 -> 2 0 1 -> 8

2 -> 2 0 1 -> -

0 -> 2 0 1 -> -

1 -> 2 0 1 -> -

7 -> 7 0 1 -> 9

0 -> 7 0 1 -> -

1 -> 7 0 1 -> -

7 7 7 2 2 2 2 2 7

0 0 0 0 4 0 0 0

1 1 3 3 3 1 1

Total number of page faults : 9

1.READ INPUT

2.FIFO

3.OPTIMAL

4.LRU

5.LFU

6.Exit

Enter your choice : 4

LRU Page Replacement Algorithm

Reference String : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page ref-> memory ->PF

7 -> 7 - - -> 1

0 -> 7 0 - -> 2

1 -> 7 0 1 -> 3

2 -> 2 0 1 -> 4

0 -> 2 0 1 -> -

3 -> 2 0 3 -> 5

0 -> 2 0 3 -> -

4 -> 4 0 3 -> 6

2 -> 4 0 2 -> 7

3 -> 4 3 2 -> 8

0 -> 0 3 2 -> 9

3 -> 0 3 2 -> -

2 -> 0 3 2 -> -

1 -> 1 3 2 -> 10

2 -> 1 3 2 -> -

0 -> 1 0 2 -> 11

1 -> 1 0 2 -> -

7 -> 1 0 7 -> 12

0 -> 1 0 7 -> -

1 -> 1 0 7 -> -

7 7 7 2 2 4 4 4 0 1 1 1

0 0 0 0 0 0 3 3 3 0 0

1 1 3 3 2 2 2 2 2 7

Total number of page faults : 12

1.READ INPUT

2.FIFO

3.OPTIMAL

4.LRU

5.LFU

6.Exit

Enter your choice : 5

LFU Page Replacement Algorithm

Reference String : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page ref -> memory -> PF

7 -> 7 - - -> 1

0 -> 7 0 - -> 2

1 -> 7 0 1 -> 3

2 -> 2 0 1 -> 4

0 -> 2 0 1 -> -

3 -> 3 0 1 -> 5

0 -> 3 0 1 -> -

4 -> 4 0 1 -> 6

2 -> 2 0 1 -> 7

3 -> 2 0 3 -> 8

0 -> 2 0 3 -> -

3 -> 2 0 3 -> -

2 -> 2 0 3 -> -

1 -> 1 0 3 -> 9

2 -> 2 0 3 -> 10

0 -> 2 0 3 -> -

1 -> 2 0 1 -> 11

7 -> 2 0 7 -> 12

0 -> 2 0 7 -> -

1 -> 2 0 1 -> 13

7 7 7 2 3 4 2 2 1 2 2 2 2

0 0 0 0 0 0 0 0 0 0 0 0

1 1 1 1 1 3 3 3 1 7 1

Total number of page faults : 13

1.READ INPUT

2.FIFO

3.OPTIMAL

4.LRU

5.LFU

6.Exit

Enter your choice : 6

Program terminated Successfully!