#include<stdio.h>

#include<time.h>

int no\_of\_frames, no\_of\_pages, frames[1000], pages[300], temp[1000], flag1, flag2, flag3, i, j, k, pos, max, faults = 0, ttime[1000];

int counter = 0;

int findLRU(int ttime[], int n);

void leastRecentlyUsed();

void optimalPageReplacement();

void init();

int main()

{

int choice,choice2,repeat=1;

double time\_taken1, time\_taken2;

clock\_t t;

do{

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter page reference string: ");

for(i = 0; i < no\_of\_pages; ++i){

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

}

printf("1 for LRU \n2 for OPTIMAL\n3. for Comparitive Study\n");

scanf("%d",&choice);

switch(choice){

case 1:

leastRecentlyUsed();

break;

case 2:

optimalPageReplacement();

break;

case 3:

printf("Do you want to use sample data?\n1 Yes \t2 No\n");

scanf("%d",&choice2);

if(choice2==1){

init();

}

t = clock();

leastRecentlyUsed();

t = clock() - t;

time\_taken1 = ((double)t)/CLOCKS\_PER\_SEC; // in seconds

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

t = clock();

optimalPageReplacement();

t = clock() - t;

time\_taken2 = ((double)t)/CLOCKS\_PER\_SEC; // in seconds

printf("\nOptimal Page Replacement Algorithm Used algorithm took %f seconds to execute\n", time\_taken2);

printf("\nLeast Recently Used Algorithm took %f seconds to execute \n", time\_taken1);

break;

}

printf("\nEnter 0 to exit\nPress any key to continue\n");

scanf("%d",&repeat);

}while(repeat!=0);

return 0;

}

void leastRecentlyUsed(){

for(i = 0; i < no\_of\_frames; ++i){

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == pages[i]){

counter++;

ttime[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == -1){

counter++;

faults++;

frames[j] = pages[i];

ttime[j] = counter;

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

pos = findLRU(ttime, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

ttime[pos] = counter;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j){

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

}

}

printf("\n\nTotal Page Faults = %d", faults);

}

int findLRU(int ttime[], int n){

int i, minimum = ttime[0], pos = 0;

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

if(ttime[i] < minimum){

minimum = ttime[i];

pos = i;

}

}

return pos;

}

void optimalPageReplacement(){

for(i = 0; i < no\_of\_frames; ++i){

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == pages[i]){

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == -1){

faults++;

frames[j] = pages[i];

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

flag3 =0;

for(j = 0; j < no\_of\_frames; ++j){

temp[j] = -1;

for(k = i + 1; k < no\_of\_pages; ++k){

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

temp[j] = k;

break;

}

}

}

for(j = 0; j < no\_of\_frames; ++j){

if(temp[j] == -1){

pos = j;

flag3 = 1;

break;

}

}

if(flag3 ==0){

max = temp[0];

pos = 0;

for(j = 1; j < no\_of\_frames; ++j){

if(temp[j] > max){

max = temp[j];

pos = j;

}

}

}

frames[pos] = pages[i];

faults++;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j){

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

}

}

printf("\n\nTotal Page Faults = %d", faults);

}

void init(){

int counter1=0, counter2,counter3;

int testArray[]= {2 ,3 ,4 ,2 ,1 ,3 ,7 ,5 ,4 ,3};

while(counter1<400){

for(counter2=0;counter2<10;counter2++){

pages[counter1+counter2]=testArray[counter2];

}

counter1= counter1 +10;

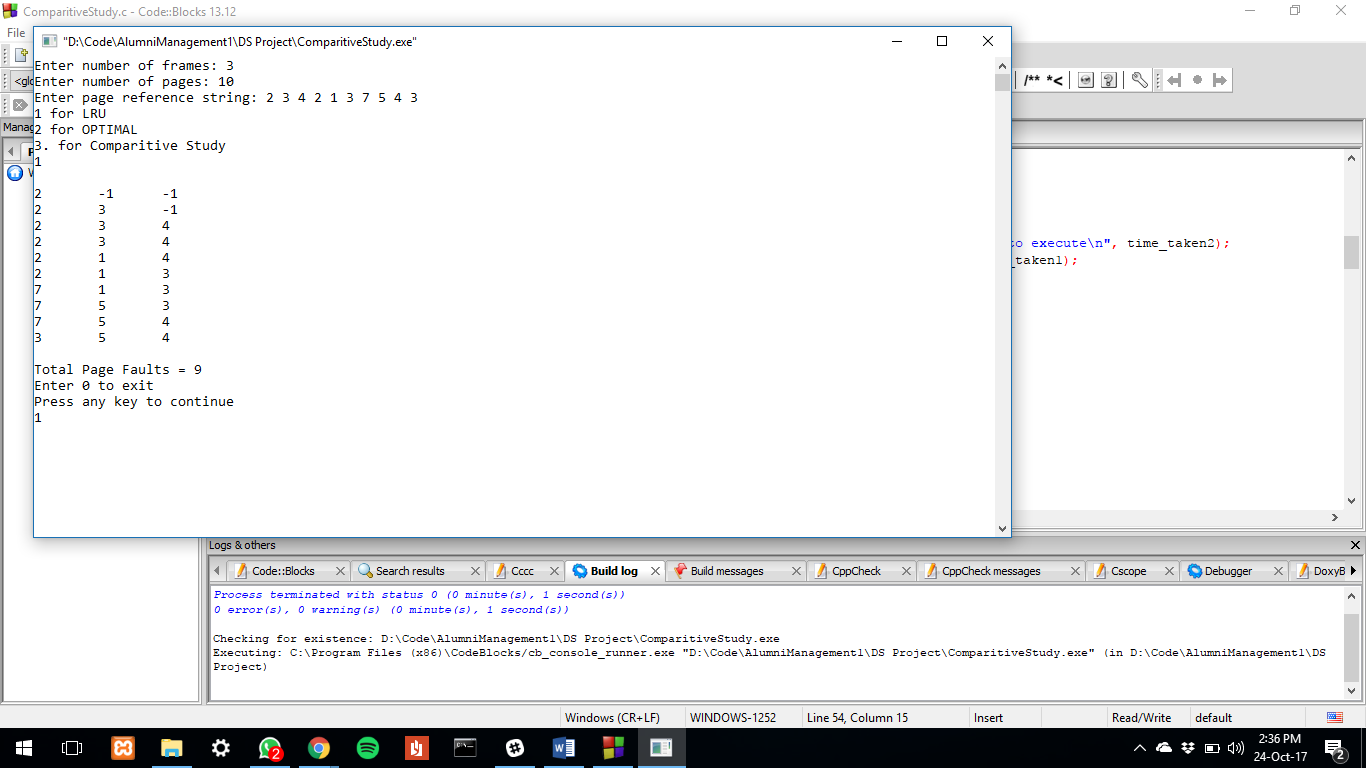
}

no\_of\_pages = 400;

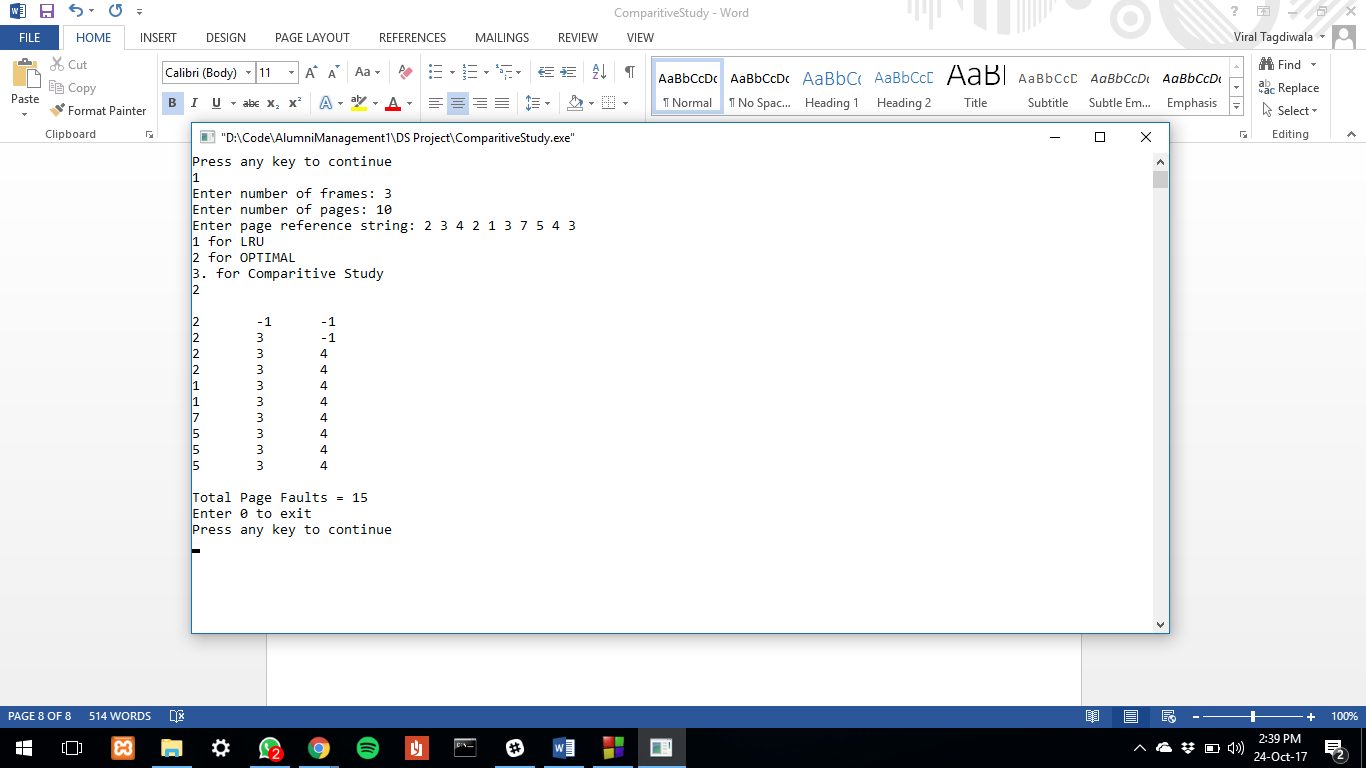
no\_of\_frames =3;

}**OUTPUT**

**LEAST RECENTLY USED**

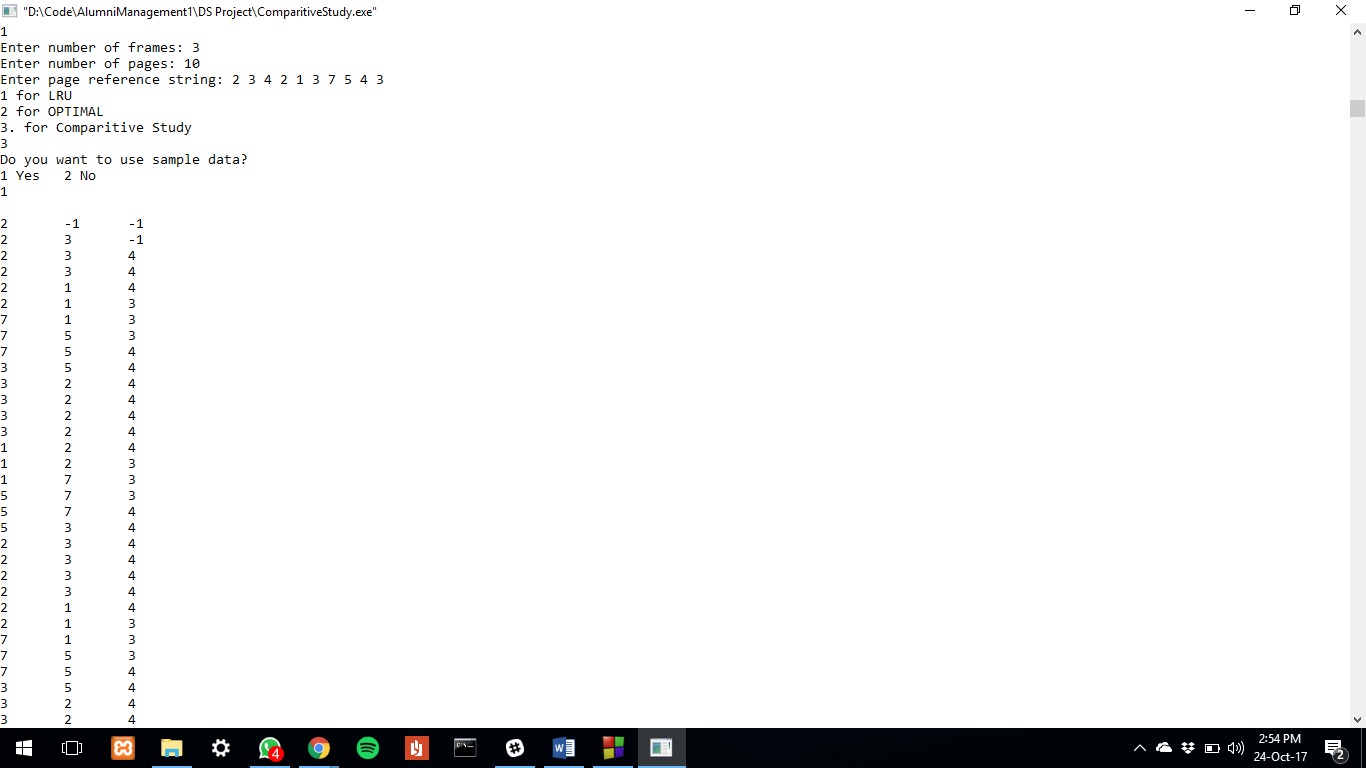


**OPTIMAL PAGE REPLACEMENT**

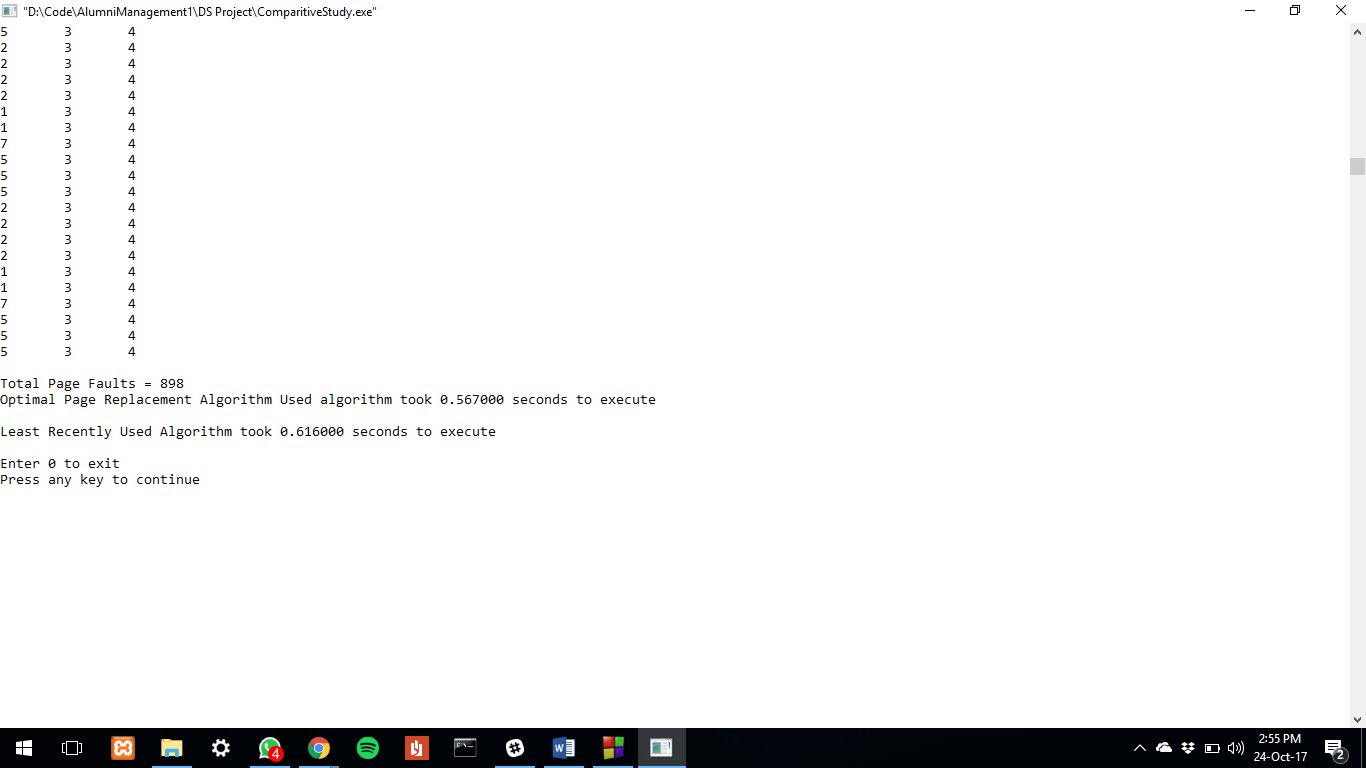


**COMPARITIVE ANALYSIS**

**WITH SAMPLE DATA**



**------------------------------------------------------------------------------------------------------------------------------------------**



**COMPARITIVE ANALYSIS**

**WITHOUT SAMPLE DATA**



**COMPARITIVE ANALYSIS – SAMPLE DATA SIZE 40,000**

