

Alexandria University

OPERATING SYSTEMS

LAB 2

Faculty of Engineering

Project Management

NAME: TOKA SHERIF ID: 5492 Group:5 sec:2

CODE part 1) Matrix Multiplication

Pseudo :

1-Void \* Mult (void \* arg ){

the function the thread calls to perform 1 elment multiplication

}

2-Void main\_element\_by\_elment ()

{

Perform array data store each row and column need to be multiplied in 1 dimmentional array

Then pass the array through a thread

}

3-Void \* mult\_join\_row (void \* arg ){

the function the thread calls to perform multiplication of row by the second matrix all

}

4-Void main\_row ()

{

Perform 1 dimensional array of elements of each row and the second matrix as 1 dimensional matrix arranged column by column to be easily multiplied

}

CODE:

#include<stdio.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

#include <time.h>

#define MAX 10

FILE \*f;

void \*mult(void\* arg)

{

int \*data = (int \*)arg;

int k = 0, i = 0;

int columnsA = data[0];

for (i = 1; i <= columnsA; i++)

k += data[i]\*data[i+columnsA];

int \*p = (int\*)malloc(sizeof(int));

\*p = k;

pthread\_exit(p);

}

int main\_elment\_by\_element ()

{

int rowsA,columnsA,rowsB,columnsB ;

int i,j,k;

f = fopen("matrix\_input.txt","r");

printf("ELEMENT BY ELEMENT METHOD\n");

printf("reading file ......\n");

fscanf(f,"%d %d", &rowsA,&columnsA);

int matA[rowsA][columnsA];

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

{

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

{

fscanf(f,"%d", &matA[i][j]);

}

}

fscanf(f,"%d %d", &rowsB,&columnsB);

int matB[rowsB][columnsB];

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

{

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

{

fscanf(f,"%d", &matB[i][j]);

}

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Matrix A\n");

// Displaying matA

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

{

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

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

printf("\n");

}

printf("Matrix B\n");

// Displaying matB

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

{

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

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

printf("\n");

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

fclose(f);

int max = rowsA\*columnsB;

clock\_t start, end;

double cpu\_time\_used;

start = clock();

pthread\_t \*threads;

threads = (pthread\_t\*)malloc(max\*sizeof(pthread\_t));

int count = 0;

int\* data = NULL;

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

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

{

//storing row and column elements in data

data = (int \*)malloc((20)\*sizeof(int));

data[0] = columnsA;

for (k = 0; k < columnsA; k++)

data[k+1] = matA[i][k];

for (k = 0; k < rowsB; k++)

data[k+columnsA+1] = matB[k][j];

//creating threads

pthread\_create(&threads[count++], NULL, mult, (void\*)(data));

}

f = fopen("matrix\_output.txt","w");

printf("RESULTANT MATRIX IS :- \n");

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

{

void \*k;

//Joining all threads and collecting return value

pthread\_join(threads[i], &k);

int \*p = (int \*)k;

printf("%d ",\*p);

fprintf(f,"%d ",\*p);

if ((i + 1) % columnsB == 0)

{

printf("\n");

fprintf(f,"\n");

}

}

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Program took %f seconds to execute \n", cpu\_time\_used);

fprintf(f,"END1 %f\n",cpu\_time\_used);

return cpu\_time\_used;

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void \* mult\_join\_row (void\* arg\_2)

{

int \*data\_2 = (int \*)arg\_2;

int k = 0, i = 0;

int shifter;

int rowsB = data\_2[0];

int columnsB = data\_2[1];

int resulted\_array[columnsB];

int \* p\_2 = (int)malloc(rowsB\*sizeof(int));

int \*l = p\_2;

for(shifter=1 ; shifter <= columnsB ; shifter++ )

{

for (i = 2 ; i <= rowsB+1 ; i++)

{

k += data\_2[i]\*data\_2[i+rowsB\*shifter];

}

//resulted\_array[shifter]=k;

//k=0;

\*p\_2 ++ = k;

k=0;

}

pthread\_exit(l);

}

int main\_row()

{

int rowsA,columnsA,rowsB,columnsB ;

// int r1=MAX,c1=MAX,r2=MAX,c2=MAX;

int i,j,k;

FILE \*f\_2 = fopen("matrix\_input.txt","r");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n");

printf("ROW BY ROW METHOD\n");

printf("reading file ......\n");

// printf("Enter no. of rows & columns for Matrix A ::");

fscanf(f\_2,"%d %d", &rowsA,&columnsA);

int matA[rowsA][columnsA];

//printf("\nEnter values to the matrix :: \n");

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

{

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

{

fscanf(f\_2,"%d", &matA[i][j]);

}

}

//printf("Enter no. of rows for Matrix B ::");

fscanf(f\_2,"%d %d", &rowsB,&columnsB);

int matB[rowsB][columnsB];

//printf("\nEnter values to the matrix :: \n");

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

{

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

{

//printf("\nEnter B[%d][%d] value :: ",i,j);

fscanf(f\_2,"%d", &matB[i][j]);

}

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Matrix A\n");

// Displaying matA

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

{

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

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

printf("\n");

}

printf("Matrix B\n");

// Displaying matB

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

{

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

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

printf("\n");

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int max = rowsA+(rowsB\*columnsB);

clock\_t start, end;

double cpu\_time\_used;

start = clock();

//declaring array of threads of size r1\*c2

pthread\_t \*threads\_2;

threads\_2 = (pthread\_t\*)malloc(rowsA\*sizeof(pthread\_t));

int count = 0;

int\* data\_2 = NULL;

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

{

//storing row and column elements in data

data\_2 = (int \*)malloc((max+2)\*sizeof(int));

data\_2[0] = rowsB;

data\_2[1] = columnsB ;

for (k = 0; k < columnsA; k++)

data\_2[k+2] = matA[i][k];

int counter = 0 ;

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

{

for (k = 0; k < (rowsB) ; k++)

{

data\_2[k+columnsA\*j+2] = matB[k][j-1];

//data1[k+c1\*j+2] = matB[k][j-1];

//data[counter+columnsA+2] = matB[k][j];

//counter++;

}

}

//creating threads

pthread\_create(&threads\_2[count++], NULL, mult\_join\_row, (void\*)(data\_2));

}

fclose(f\_2);

//f\_2=fopen("matrix\_output.txt","a");

printf("RESULTANT MATRIX IS :- \n");

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

{

void \*k\_2;

//Joining all threads and collecting return value

pthread\_join(threads\_2[i], &k\_2);

int \*p\_2 = (int \*)k\_2;

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

{

fprintf(f,"%d ",\*p\_2);

printf("%d ",\*p\_2);

\*p\_2++;

}

printf("\n");

fprintf(f,"\n");

}

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Program took %f seconds to execute \n", cpu\_time\_used);

fprintf(f,"END2 %f",cpu\_time\_used);

fclose(f\_2);

fclose(f);

return cpu\_time\_used;

}

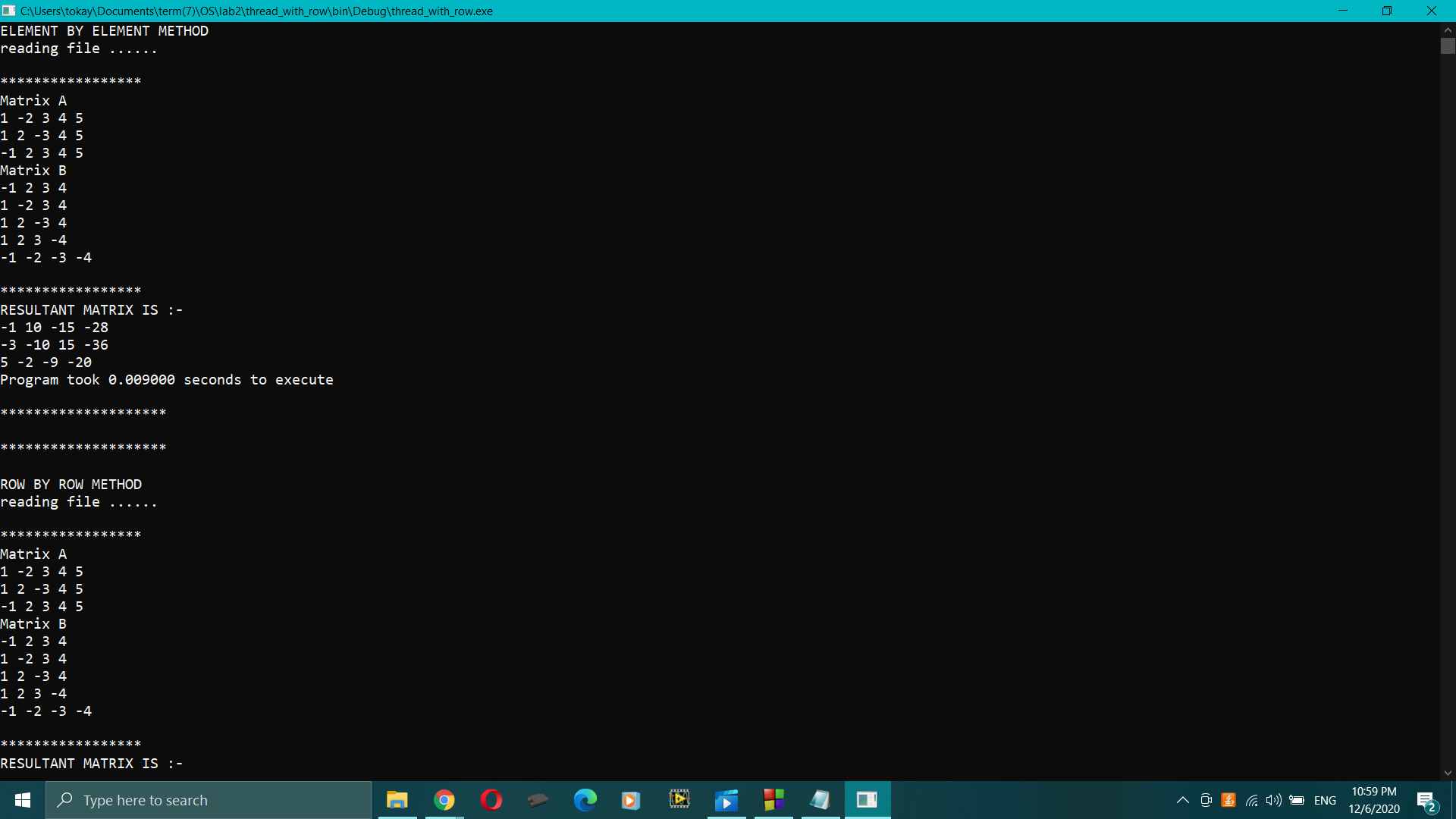
int main()

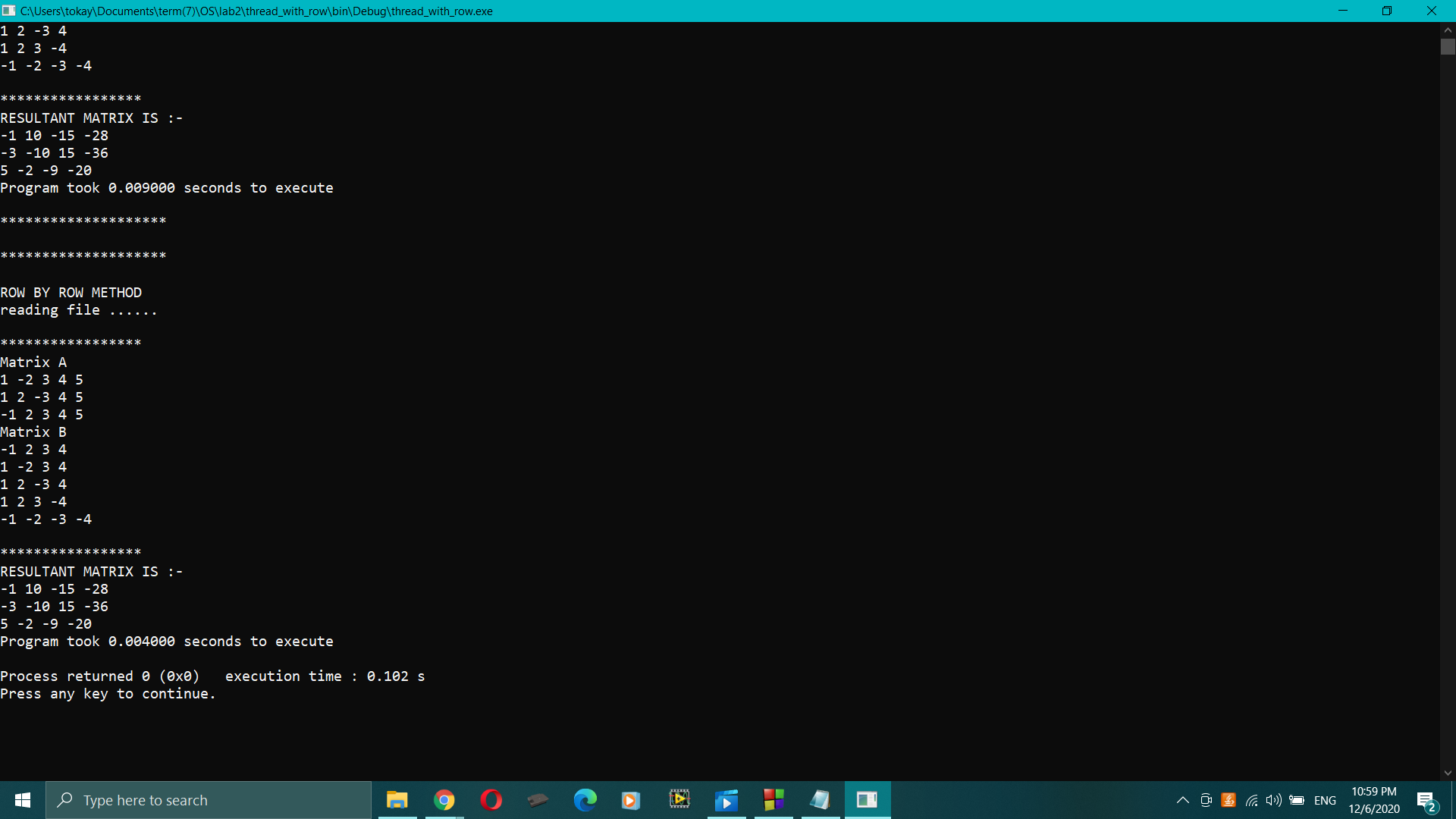
{

int first\_time\_elapsed = main\_elment\_by\_element() ;

int secind\_time\_elapsed = main\_row();

}





CODE part 2) Merge Sort

Pseudo :

struct struct\_list{

int low ;

int high;

}

Merge{

After partiotioning it merge 2 sub-lists to be ordered

}

Void\* mergeSort(void\* arg )

{

Create 2 structs

Create 2 threads

First thread and struct passed to pthread\_create for left list

Join returned values

Second thread and struct passed to pthread\_create for right list

Join returned values

Merge all

}

CODE:

/\* C program for Merge Sort \*/

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

struct struct\_list {

int low;

int high;

};

int \*arr;

void merge( int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void\* mergeSort(void\* arg)

{

//printf("entred thread\n ");

struct struct\_list \*data;

data = (struct struct\_list\*) malloc( sizeof(struct struct\_list));

data = (struct struct\_list\*) arg ;

//printf("low\_arg :: %d\n",data->low);

//printf("high\_arg :: %d\n",data->high);

int l = data->low;

int r= data->high;

if (l < r) {

int mid = l + (r - l) / 2;

struct struct\_list list\_left;

list\_left.low=l;

list\_left.high = mid;

//printf("low\_leftthread :: %d\n",list\_left.low);

//printf("high\_leftthread :: %d\n",list\_left.high);

//printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

struct struct\_list list\_right;

list\_right.low = mid+1;

list\_right.high= r;

//printf("low\_rightthread :: %d\n",list\_right.low);

//printf("high\_rightthread :: %d\n",list\_right.high);

//printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

pthread\_t thread\_left;

pthread\_create(&thread\_left , NULL , mergeSort , (void\*) &list\_left );

pthread\_join(thread\_left,NULL);

pthread\_t thread\_right ;

pthread\_create(&thread\_right ,NULL , mergeSort , (void \*) &list\_right);

pthread\_join(thread\_right,NULL);

merge(l, mid, r);

}

}

void printArray(int A[], int size)

{

FILE \* f;

f = fopen("output\_merge.txt","w");

int i;

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

{

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

fprintf(f,"%d",A[i]);

}

printf("\n");

}

int main()

{

FILE\* f ;

f = fopen("input\_merge.txt","r");

int arr\_size,i;

fscanf(f,"%d",&arr\_size);

arr = (int \*)malloc((arr\_size)\*sizeof(int));

for(i=0 ;i<arr\_size;i++)

fscanf(f,"%d",&arr[i]);

printf("THE ARRAY FROM FILE::\n");

for(i=0 ; i< arr\_size ; i++)

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

printf("\n\*\*\*\*\*\*\*\*\*\*\n");

struct struct\_list \* list;

list = (struct struct\_list\*) malloc( sizeof(struct struct\_list));

list->low = 0;

list->high=arr\_size-1;

// printf("mainArray\_low:: %d\n",list->low);

//printf("mainArray\_high :: %d\n",list->high);

// printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

pthread\_t thread;

pthread\_create(&thread , NULL , mergeSort , (void\*) list );

pthread\_join(thread,NULL);

//printf("returned from merge\n");

fclose(f);

printf("\n\*\*\*\*\*\*\*\*\*\*\n");

printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}

