#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#include <stdbool.h>

#include <time.h>

#define numThreads 2048

int array[numThreads\*2];

int dup\_array[numThreads\*2];

typedef struct

{

int left;

int right;

}parameters;

parameters \*para[numThreads];

int arraySize=0;

pthread\_barrier\_t barr; // barrier variable

pthread\_mutex\_t lock; // define a mutual exclusion lock for threads.

// read array data from indata.txt .

void readArray(){

int i=0;

time\_t t;

srand((unsigned) time(&t));

arraySize = rand() % 1001;

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

{

array[i] = rand() % 1001;

}

int ind =0;

for(ind=0;ind<arraySize;ind++)

{

dup\_array[ind] = array[ind];

}

arraySize=i;

printf("arraySize = %d \n",arraySize);

}

bool checkArraySize(void){

int n=arraySize;

int i=0;

while(n!=1)

{

i++;

n=n>>1;

}

n=arraySize;

if(((n>>i)<<i)==n)

{

return true;

}

else

{

long long req\_size = (long long)1 << (i+1);

int ind = 0;

for(ind = arraySize;ind < req\_size;ind++)

{

array[ind] = INT\_MAX;

}

arraySize = req\_size;

int itr = 0;

return true;

}

}

void \*mergeSort(void \*para)

{

int left,right,mid,i,j,k;

int \*leftArray,\*rightArray;

parameters \*para\_=para;

left=para\_->left;

right=para\_->right;

mid=(left+right)/2;

leftArray=(int \*)calloc(sizeof(int),right-left+1);

rightArray=(int \*)calloc(sizeof(int),right-left+1);

//printf("inthread: %d %d %d \n",left,right,mid);

for(i=left;i<=mid;i++){

leftArray[i-left]=array[i];

}

for(j=mid+1;j<=right;j++){

rightArray[j-mid-1]=array[j];

}

i=left; j=mid+1; k=left;

while(i<=mid&&j<=right){

if(leftArray[i-left]<=rightArray[j-mid-1]){

array[k]=leftArray[i-left];

i++; k++;

}else{

array[k]=rightArray[j-mid-1];

j++; k++;

}

}

while(i<=mid){

array[k]=leftArray[i-left];

i++; k++;

}

while(j<=right){

array[k]=rightArray[j-mid-1];

j++; k++;

}

free(leftArray);

free(rightArray);

//for(i=left;i<=right;i++)printf("%d ",array[i]);

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

//pthread\_mutex\_lock(&lock);

pthread\_barrier\_wait(&barr);

//pthread\_mutex\_unlock(&lock);

//pthread\_exit(NULL);

}

int main(){

int i,j,k;

pthread\_t tid[numThreads]; // to store the id of each threads.

pthread\_mutex\_init(&lock,NULL); // initiate the mutual exclusion lock.

readArray();

if(checkArraySize())

{

i=arraySize/2; // number of threads in this merge layer

int height = 1;

clock\_t start\_time,end\_time;

start\_time = clock();

while(1)

{

j=arraySize/i; // number of elements to be sorted in single thread

printf("Layer %d : Number of threads = %d , Number of elements in each thread = %d \n",height,i,j);

if(pthread\_barrier\_init(&barr,NULL,i+1)) // initiate barrier with a count of i+1;

{

printf("Could not create a barrier\n");

return -1;

}

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

{

para[k]=(parameters\*)malloc(sizeof(parameters));

para[k]->left=k\*j;

para[k]->right=(k+1)\*j-1;

if(pthread\_create(&tid[k],NULL,mergeSort,para[k]))

{

printf("Could not create thread\n");

return -1;

}

}

pthread\_barrier\_wait(&barr);// wait for all threads being finished.

pthread\_barrier\_destroy(&barr);

for(k=0;k<i;k++)free(para[k]);

if(i==1)break; // merge sort has been finished if i==1

else i=i/2; // merge sort for next layer

height++;

}

end\_time = clock();

printf("Time taken by merge sort using mulithreaded programming : %f \n",((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC);

}

//printf("sorted array: ");

//for(i=0;i<arraySize && array[i]!=INT\_MAX;i++){

// printf("%d ",array[i]);

//}

printf("\n");

pthread\_mutex\_destroy(&lock); // destroy the mutual exclusion lock.

}