#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <dirent.h>

#include <string.h>

#include <pthread.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/mman.h>

pthread\_mutex\_t mutexA = PTHREAD\_MUTEX\_INITIALIZER;

typedef struct thread\_args

{

int thread\_num;

int num\_threads;

int num\_files;

int result;

char \*d;

} thread\_args;

void \*compareFiles(void \*ptr) {

struct thread\_args \*args = (thread\_args \*) ptr;

int num\_threads = args->num\_threads;

int thread\_num = args->thread\_num;

int num\_files = args->num\_files;

char \*d = args->d;

int \*unsorted\_file\_memory;

int \*sorted\_file\_memory;

struct stat sb;

struct stat sb2;

for (int itr=thread\_num; itr<num\_files;itr += num\_threads) {

char unsorted\_filepath[5000];

sprintf(unsorted\_filepath, "%s/unsorted\_%d.bin", d, itr);

int fd = open(unsorted\_filepath, O\_RDONLY);

if (fd == -1) {

fprintf(stderr, "Unsorted file %s doesn't exist, returning...\n", unsorted\_filepath);

args->result = -1;

pthread\_exit (NULL);

}

if (stat(unsorted\_filepath, &sb) == -1) {

fprintf(stderr, "Could not get file stat for %s, exiting...\n", unsorted\_filepath);

args->result = -1;

pthread\_exit (NULL);

}

unsorted\_file\_memory = mmap(NULL, sb.st\_size, PROT\_READ, MAP\_PRIVATE, fd, 0);

char sorted\_filepath[5000];

sprintf(sorted\_filepath, "%s/sorted/sorted\_%d.bin", d, itr);

// printf("Sorted file : %s\n", sorted\_filepath);

int fd2 = open(sorted\_filepath, O\_RDONLY);

if (fd2 == -1) {

fprintf(stderr, "Sorted file %s doesn't exist, returning...\n", sorted\_filepath);

args->result = -1;

munmap(unsorted\_file\_memory, sb.st\_size);

pthread\_exit (NULL);

}

if (stat(sorted\_filepath, &sb2) == -1) {

fprintf(stderr, "Could not get file stat for %s, exiting...\n", sorted\_filepath);

args->result = -1;

munmap(unsorted\_file\_memory, sb.st\_size);

pthread\_exit (NULL);

}

sorted\_file\_memory = mmap(NULL, sb2.st\_size, PROT\_READ, MAP\_PRIVATE, fd2, 0);

// printf("sorted memory size = %ld\n", sb2.st\_size);

// check if all sorted numbers are there in the unsorted numbers

for (int i=0; i<sb.st\_size/sizeof(int); i++) {

int found = 0;

for (int j=0; j<sb2.st\_size/sizeof(int); j++) {

if (unsorted\_file\_memory[i] == sorted\_file\_memory[j]){

//printf("match found\n");

found = 1;

continue;

}

}

if (found == 0) {

//printf("mismatch occured at i=%d\n",i);

fprintf(stderr, "Could not match the sorted numbers with the unsorted numbers\n");

args->result = -1;

munmap(unsorted\_file\_memory, sb.st\_size);

munmap(sorted\_file\_memory, sb2.st\_size);

pthread\_exit (NULL);

}

}

// check if the numbers are actually sorted

int previous\_number = sorted\_file\_memory[0];

for (int i=1; i<sb.st\_size/sizeof(int); i++) {

int current\_number = sorted\_file\_memory[i];

if (current\_number < previous\_number) {

fprintf(stderr, "Files were not sorted!\n");

args->result = -1;

munmap(unsorted\_file\_memory, sb.st\_size);

munmap(sorted\_file\_memory, sb2.st\_size);

pthread\_exit (NULL);

}

}

}

munmap(unsorted\_file\_memory, sb.st\_size);

munmap(sorted\_file\_memory, sb2.st\_size);

pthread\_exit(NULL);

}

int main(int argc, char \*\*argv) {

//FILECHECKER TEST: try with 0 threads

char \*q = argv[2];

int res;

res = strcmp(q,"0");

if(res == 0) {

fprintf(stderr,"Zero threads provided, exiting the code..\n");

return -1;

}

char \*d = argv[1];

char \*ptr;

if (d == NULL) {

fprintf(stderr, "No source folder provided, exiting the code...\n");

return -1;

}

if (argc < 3) {

fprintf(stderr, "Not all arguments passed, Follow the pattern\n./filechecker <path> <number threads>\nclosing the program...");

return -1;

}

long t = strtol(argv[2], &ptr, 10);

int source\_dir\_exist = access(d, F\_OK);

if (source\_dir\_exist == -1) {

fprintf(stderr, "Could not locate the source folder, exiting the code...\n");

return -1;

}

int source\_dir\_permission = access(d, R\_OK);

if (source\_dir\_permission == -1) {

fprintf(stderr, "Do not have read permission to folder, exiting the code...\n");

return -1;

}

// read from source

// check if files are there, return error if no files

DIR \*dir\_open = opendir(d);

struct dirent\* in\_file;

int file\_count = 0;

while ((in\_file = readdir(dir\_open))) {

if (!strcmp (in\_file->d\_name, "."))

continue;

if (!strcmp (in\_file->d\_name, ".."))

continue;

if (!strcmp (in\_file->d\_name, "sorted"))

continue;

file\_count += 1;

}

if (file\_count == 0){

fprintf(stderr, "folder is empty, exiting the code...\n");

return -1;

}

pthread\_t w1[t];

thread\_args\* ta[t];

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

struct thread\_args \*args = malloc(sizeof(thread\_args));

args->thread\_num = i;

args->num\_threads = t;

args->d = d;

args->num\_files = file\_count;

ta[i] = args;

pthread\_create(&w1[i], NULL, &compareFiles, args);

}

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

pthread\_join(w1[i], NULL);

}

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

int result = ta[i]->result;

if (result == -1) {

fprintf(stderr, "Error occured, exiting the code...\n");

return -1;

}

}

printf("All looks good\n");

return 0;

}