#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <pthread.h>

#include <openssl/evp.h>

#define MAX\_THREADS 6

#define BUF\_SIZE 1024

struct ThreadArgs {

const char \*inputFile;

const char \*outputFile;

};

void handleErrors(void) {

fprintf(stderr, "Error occurred\n");

exit(1);

}

void\* encryptFile(void\* arg) {

struct ThreadArgs \*threadArgs = (struct ThreadArgs \*)arg;

const char \*key = "your\_encryption\_key"; // Change this to a secure method of key storage

const char \*inputFile = threadArgs->inputFile;

const char \*outputFile = threadArgs->outputFile;

// ... (rest of the encryption process)

return NULL;

}

int main() {

const char \*inputFolder = "input\_folder/";

const char \*outputFolder = "output\_folder/";

// ... (open directory and initialize)

if ((dir = opendir(inputFolder)) != NULL) {

pthread\_t threads[MAX\_THREADS];

int threadIndex = 0;

while ((ent = readdir(dir)) != NULL) {

if (ent->d\_type == DT\_REG) { // Regular file

if (threadIndex < MAX\_THREADS) {

struct ThreadArgs threadArgs;

snprintf(threadArgs.inputFile, sizeof(threadArgs.inputFile), "%s%s", inputFolder, ent->d\_name);

snprintf(threadArgs.outputFile, sizeof(threadArgs.outputFile), "%s%s.enc", outputFolder, ent->d\_name);

pthread\_create(&threads[threadIndex], NULL, encryptFile, &threadArgs);

threadIndex++;

}

}

}

// Wait for all threads to finish

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

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

}

closedir(dir);

} else {

perror("Error opening directory");

return 1;

}

printf("All files encrypted successfully!\n");

return 0;

}