Sure, if you have an external executable called `aes\_encrypt` that performs the encryption and you want to use `fork` and `execve` to run it from your code, here's how you can modify your existing code:

```c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <pthread.h>

#include <unistd.h> // For fork and execve

#include <sys/wait.h> // For waitpid

#define MAX\_THREADS 6

#define BUF\_SIZE 1024

void\* encryptFile(void\* arg) {

// ... (same as before)

// Build the command to run the external encryption executable

char command[256];

snprintf(command, sizeof(command), "./aes\_encrypt %s %s", inputFile, outputFile);

// Use fork and execve to run the encryption executable

pid\_t pid = fork();

if (pid == 0) { // Child process

char \*argv[] = {"./aes\_encrypt", inputFile, outputFile, NULL};

execve(argv[0], argv, NULL);

exit(1); // In case execve fails

} else if (pid > 0) { // Parent process

int status;

waitpid(pid, &status, 0);

if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {

printf("File %s encrypted successfully!\n", filename);

} else {

printf("Encryption failed for file %s\n", filename);

}

} else { // Fork error

perror("Fork error");

}

return NULL;

}

int main() {

// ... (same as before)

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

pthread\_create(&threads[threadIndex], NULL, encryptFile, ent->d\_name);

threadIndex++;

if (threadIndex >= MAX\_THREADS) {

for (int i = 0; i < MAX\_THREADS; i++) {

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

}

threadIndex = 0;

}

}

}

// Wait for remaining 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;

}

```

In this modified version of the code, the changes include:

1. The command to run the external `aes\_encrypt` executable is built using `snprintf`.

2. `fork` is used to create a child process.

3. In the child process, `execve` is used to replace the current process with the external executable. If `execve` fails, the child process exits with an error code.

4. In the parent process, `waitpid` is used to wait for the child process to complete.

5. Depending on the exit status of the child process, a success or failure message is printed.

Make sure you have the `aes\_encrypt` executable in the same directory as your program for this to work.

Certainly! Here's the modified version of your code that uses `execve` directly in the `encryptFile` function:

```c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <pthread.h>

#include <unistd.h> // For fork and execve

#include <sys/wait.h> // For waitpid

#define MAX\_THREADS 6

#define BUF\_SIZE 1024

void\* encryptFile(void\* arg) {

// ... (same as before)

// Build the arguments array for execve

char \*argv[] = {"./aes\_encrypt", (char \*)inputFile, (char \*)outputFile, NULL};

execve(argv[0], argv, NULL);

perror("execve"); // This line will be reached only if execve fails

return NULL;

}

int main() {

// ... (same as before)

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

pthread\_create(&threads[threadIndex], NULL, encryptFile, ent->d\_name);

threadIndex++;

if (threadIndex >= MAX\_THREADS) {

for (int i = 0; i < MAX\_THREADS; i++) {

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

}

threadIndex = 0;

}

}

}

// Wait for remaining 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;

}

```

In this version of the code, the `encryptFile` function now directly calls `execve` to execute the `aes\_encrypt` executable with the appropriate arguments. If `execve` fails, an error message is printed using `perror`.

Please ensure that you have the `aes\_encrypt` executable in the same directory as your program and that it's properly configured to accept the input and output file paths as command-line arguments. Also, be aware of the potential complexities of using `execve` in a multithreaded environment and make sure your design is sound.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <pthread.h>

#include <sys/wait.h>

#include <unistd.h>

#define MAX\_THREADS 6

#define BUF\_SIZE 1024

int main() {

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

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

DIR \*dir;

struct dirent \*ent;

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

int threadCount = 0;

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

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

if (threadCount >= MAX\_THREADS) {

// Wait for the spawned processes to finish before creating more

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

wait(NULL);

}

threadCount = 0;

}

pid\_t pid = fork();

if (pid < 0) {

perror("Fork error");

return 1;

} else if (pid == 0) {

// Child process

char inputFile[256];

char outputFile[256];

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

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

execl("./aes\_encrypt", "./aes\_encrypt", inputFile, outputFile, (char \*)NULL);

perror("Exec error");

exit(1);

} else {

// Parent process

threadCount++;

}

}

}

// Wait for remaining child processes to finish

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

wait(NULL);

}

closedir(dir);

} else {

perror("Error opening directory");

return 1;

}

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

return 0;

}

Certainly! If your `aes\_encrypt` executable requires both a filename and an external key path as command-line arguments, you can modify the code as follows:

```c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <pthread.h>

#include <sys/wait.h>

#include <unistd.h>

#define MAX\_THREADS 6

#define BUF\_SIZE 1024

int main() {

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

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

const char \*keyPath = "path\_to\_key\_file";

DIR \*dir;

struct dirent \*ent;

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

int threadCount = 0;

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

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

if (threadCount >= MAX\_THREADS) {

// Wait for the spawned processes to finish before creating more

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

wait(NULL);

}

threadCount = 0;

}

pid\_t pid = fork();

if (pid < 0) {

perror("Fork error");

return 1;

} else if (pid == 0) {

// Child process

char \*args[] = {

"./aes\_encrypt",

(char \*)ent->d\_name,

keyPath,

NULL

};

char inputFile[256];

char outputFile[256];

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

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

char \*envp[] = { NULL };

if (execve("./aes\_encrypt", args, envp) == -1) {

perror("Execve error");

exit(1);

}

} else {

// Parent process

threadCount++;

}

}

}

// Wait for remaining child processes to finish

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

wait(NULL);

}

closedir(dir);

} else {

perror("Error opening directory");

return 1;

}

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

return 0;

}

```

In this version of the code, I've added a `keyPath` variable to store the path to the external key file. The `keyPath` is then passed as an argument to the `aes\_encrypt` executable along with the filename.

Please make sure that your `aes\_encrypt` executable expects the filename as the first argument and the key path as the second argument. Adjust the argument order in the `args` array if necessary. Also, ensure that the `aes\_encrypt` executable is capable of reading the key file from the provided path.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <pthread.h>

#include <sys/wait.h>

#include <unistd.h>

#define MAX\_THREADS 6

#define BUF\_SIZE 1024

int main() {

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

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

const char \*keyPath = "path\_to\_key\_file";

DIR \*dir;

struct dirent \*ent;

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

int threadCount = 0;

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

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

if (threadCount >= MAX\_THREADS) {

// Wait for the spawned processes to finish before creating more

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

wait(NULL);

}

threadCount = 0;

}

pid\_t pid = fork();

if (pid < 0) {

perror("Fork error");

return 1;

} else if (pid == 0) {

// Child process

char \*args[] = {

"./aes\_encrypt",

(char \*)ent->d\_name,

keyPath,

NULL

};

char inputFile[256];

char outputFile[256];

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

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

char \*envp[] = { NULL };

if (execve("./aes\_encrypt", args, envp) == -1) {

perror("Execve error");

exit(1);

}

} else {

// Parent process

threadCount++;

}

}

}

// Wait for remaining child processes to finish

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

wait(NULL);

}

closedir(dir);

} else {

perror("Error opening directory");

return 1;

}

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

return 0;

}