# *myfind Protokoll*

## Überblick

Das Programm sucht nach Dateien in einem Verzeichnis, wahlweise rekursiv und ohne Berücksichtigung von Groß- und Kleinschreibung. Für jede Datei, die gesucht wird, startet das Programm einen eigenen Kindprozess mit fork(). Dadurch laufen mehrere Suchen gleichzeitig. Der Elternprozess sammelt später die Ergebnisse und gibt sie in geordneter Reihenfolge aus.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

specifik@DESKTOP-H0SHFAU:~/projects/myfind$ make
g++ -Wall -Wextra -std=c++17 -pedantic -g -c myfind.cpp -o myfind.o
g++  myfind.o -o myfind
specifik@DESKTOP-H0SHFAU:~/projects/myfind$ ./myfind ./ test test.doc test.txt
7162: test: "/home/specifik/projects/myfind/./test"
7163: test.doc: "/home/specifik/projects/myfind/./test.doc"
7164: test.txt: "/home/specifik/projects/myfind/./test.txt"
specifik@DESKTOP-H0SHFAU:~/projects/myfind$ ./myfind ./ -R test
7569: test: "/home/specifik/projects/myfind/./test"
specifik@DESKTOP-H0SHFAU:~/projects/myfind$ ./myfind ./ -i test
7640: test: "/home/specifik/projects/myfind/./test"
specifik@DESKTOP-H0SHFAU:~/projects/myfind$ ./myfind ./ -R -i test
7736: test: "/home/specifik/projects/myfind/./test"
specifik@DESKTOP-H0SHFAU:~/projects/myfind$ make clean
rm -f myfind.o myfind
specifik@DESKTOP-H0SHFAU:~/projects/myfind$
```

## Parallelisierung mit fork()

Durch fork() wird für jede zu suchende Datei ein eigener Kindprozess erstellt. Diese Prozesse laufen parallel, was die Suche bei mehreren Dateien beschleunigt. Jeder Prozess sucht nur nach einer bestimmten Datei, unabhängig von den anderen.

```
for (const auto& filename : filenames) {
    // Create a temporary file for each child process
    string tempFile = "/tmp/myfind_" + filename + "_" + to_string(getpid()) + ".txt";
    tempFiles.push_back(tempFile);
    int fd = open(tempFile.c_str(), O_CREAT | O_WRONLY, 0600);

    pid_t pid = fork();
    if (pid == 0) {
        searchFile(searchpath, filename, recursive, caseInsensitive, fd);
        exit(EXIT_SUCCESS);
    } else if (pid > 0) {
        childPids.push_back(pid);
        close(fd);
    } else {
        cerr << "Fork failed" << endl;
        return EXIT_FAILURE;
    }
}
```

## Temporäre Dateien

Statt direkt in stdout zu schreiben, leitet jeder Kindprozess seine Ausgabe in eine eigene temporäre Datei um. Diese Dateien werden später vom Elternprozess nacheinander gelesen und ausgegeben, wodurch die Ergebnisse in geordneter Reihenfolge erscheinen.

```
    // Redirect stdout to the provided file descriptor
    dup2(fd, STDOUT_FILENO);
    close(fd);
```

## Warten auf Prozesse mit waitpid()

Der Elternprozess wartet mit waitpid() darauf, dass alle Kindprozesse abgeschlossen sind, bevor er die temporären Dateien liest. Das verhindert, dass "Zombie"-Prozesse entstehen und stellt sicher, dass alle Suchergebnisse vorliegen, bevor die Ausgabe erfolgt.

```
    // Parent process waits for all child processes to terminate
    // This prevents zombie processes by reading the exit status of each child process
    for (pid_t pid : childPids) {
        waitpid(pid, nullptr, 0);
    }
```