myFind

We decided to utilize additional processes for search inside of subdirectories. And this will take place only by recursive search. Why so? When we have a non recursive search the overhead of creating and managing the synchronization of processes for a search inside of a single directory is too big, so one for loop is more that enough. We simply store all file names in a hashset and in that case we do not need nested for loops we just check if current file is in the hashset and if yes then send to the message queue.

```
### FileSearcher.h

for (const auto &entry: fs::directory_iterator(_path)) {
if (entry.is_directory()) _dir_search_strategy->search(entry.path(), shared_from_this());
if (_case_search_strategy->file_found(entry)) {
std::string message = std::to_string((int) getpid()) + ": " +
entry.path().filename().string() + ": " + entry.path().string() + "\n";
_mq->send_message(message);
}
}
```

Yes, for synchronization and communication of processes we use a message queue to send found file names from child processes of the main child process and accept them in the loop of the main process.

```
### MessageQueue.h

message_buffer buffer;

while (true) {
msgrcv(mq_id, &buffer, sizeof(buffer.message_text), MESSAGE_TYPE, 0);
std::string message(buffer.message_text);
if (message == END_OF_MESSAGES_MARKER) break;
messages.emplace_back(std::move(message));
}

msgctl(mq_id, IPC_RMID, NULL);
```

We also send a final message from the main child process so that the main process terminates waiting for new messages when all children of all children and so on are finished. **So the END_OF_MESSAGE is sent when all children processes are ready!**

```
### main.h

if (search_pid == 1) return EXIT_FAILURE;
// main child process which launches search and other processes for subdirectories
else if (search_pid == 0) {
file_searcher->search();
mq->send_message(mq->END_OF_MESSAGES_MARKER);
return EXIT_SUCCESS;
}
...
snippet from above with while(true)
```

All search processes store pids of their children in a vector and then wait for them all to finish in a method **wait_for_all()** after the search in their subdirectories is done. And search and waiting are done the process finishes his job with **exit(0)** and his parent process then deletes it from the vector when it sees that it is finished and waits for others.

```
void wait_for_all() {
int status;
while (!running_processes.empty()) {
pid_t done_process = wait(&status);
if (done_process != -1) running_processes.erase(done_process);
}
}
```