Concurrent Programming - Concurrency Project

Thor Skjold Haagensen.221190.Thhaa16@student.sdu.dk .
Supervisor: Fabrizio Montesi
DM519

May 8, 2017

Contents

1 Methodology

The program implements the findAll and the findAny method. Both methods work by using ideas from the producer / consumer concept, and they share the same procedure when processing data. I will be explaining findAll and then point out where findAny differ.

findAll is called with a path to a directory, it starts by instantiating a integer to get available processors, it then instantiates a executor service with the processor count, afterwards it creates two arraylists, resultset and set. resultset is used to gather objects of the type ResultClass which is a class that implements the interface Result, this list is returned when the program has finished and the method findAny does not make use of this. The arraylist set is a list containing futures of results. findAll then calls the method visit with the path, the arraylist set, executor and a new object of the nested consumer class (This class will be explained later). Instantiating this local to the method makes it possible to reuse the visit method for findAny.

visit actually takes a function but since consumer class implements callable and has the same return type e.g. Result, it is allowed to pass it. The visit method recursively traverses a dictionary, if it meets a subpath it calls visit with that as path, but when meeting a .txt file, it submits to the executor the function / object consumer with its found path, this is added to the list ”set” (which contains futures of results).

The Consumer class as mentioned implements Callable¡ Result ¿ and takes a path when instantiated. it inherits a call() method which creates a arraylist of all the numbers in the file and then iterates over the list to find the largest number, once found it creates a ResultClass object with the max number and the path to the file.

The ConsumerAny class works almost the same way, but aside from the path it also needs two integers n and min when instantiated. These are the result criteria which is tested against when iterating over integers in a file, at the same time it also tests whether its thread has been canceled, if so it just returns null. This method also returns an object of the type ResultClass class.

Going back to findAll, once it has called visit the method will iterate over the list of futures ”set” and add each result to the final list resultset which when done will be returned post executor shutdown.

The findAny method instead of calling visit with the cunsomer as a lambda, wraps ConsumerAny in a function f with it’s extra variables (this is done since the lambda in the visit call cant take more than one argument). When processing the result the method still iterates over the list set but since it only needs one result which meets the criteria it will get the first it can and call executor.shutdownNow(), canceling all remaining threads and afterwards returning the result.

When designing the program i thought about which tasks would be the ones requiring most computer power and which would take the most time, I concluded that recursively traversing a dictionary and a arraylist would be somewhere around n time (where n is the number of elements in the directory / subdirectories) which compared to going through a lot of files with an unknown number of integers is relative cheap, therefore i focused on making the latter part concurrent. This as described is achieved with an executor service that adds results to a list of futures. Using an executor was chosen because there will be created a function for every .txt file in the directories which would be a lot to manually keep track of therefore i let the executor do the work for me,

it has the bonus of also optimizing the code so it will run faster for larger directories, the futures was chosen as an easy way to keep thread safety when returning and using the results, it also guarantees that all files has been processes before returning the final list of results.
I felt that since findAny and findAll worked sort of similarly, it made sense to reuse the visit method, originally the visit method would add .txt file's paths to a blockingdeque which the consumers would take from, but that was discovered to be a unnecessary and potentially costly operation, since scenarios where the threads have to wait excessive time could easily be thought up. Therefore I chose to have the visit method pass the paths directly to the executor, this also guarantees that when the list of futures are empty, all the .txt files in the subdirectories have been processed.

2 Advantages

During development a big emphasis was put on splitting every operation into its own method for the purpose of having good readability, I would argue that I was fairly successful in achieving this goal but of course I was the one who wrote it so i might not be the best judge of the matter.

On the relative small test directories that was given with with the exam the program will run almost as slow as a non concurrent program because of the cost of creating and maintaining threads, but with larger directories the program will perform much better and I am fairly satisfied with the speed.
The speed of the program would scale well with the number of cores, since the biggest workload is in the multi threaded part of the program.

Additions to the program could be implemented, and I can see the program performing other tasks of similar approach e.g. going through directories and finding information in files, be it strings, bits or anything else.

3 Limitations

In using the visit method to direct the workload to the executor we potentially miss out on some concurrency, better results might be achieved by having the visit method work concurrently and then using some sort of wait / notify on the consumers, but this approach could potentially result in some unwanted wait time when threads having to wait on locks etc. Which is something i tried to eliminate with my approach to the problem.
Error handling could probably in some cases have been done better.

As a conclusion the program runs satisfactory and I think i handled concurrency in a nice way, i would have liked to have a good implementation of the stats method but unfortunately that wasn't possible.