CSC 412 – Operating Systems

Programming Assignment 05, Spring 2022

Tuesday, April 5th, 2022

Due date: Friday, April 15th, 11:59spm.

1 What This Assignment Is About

1.1 Objectives

The objectives of this assignment are for you to

- Get more experience with process creation in Unix;
- Develop a multithreaded version of an application;
- Implement a script that continuously monitors a "drop folder."

This is an individual assignment.

1.2 Handout

The handouts for this assignment are:

- this document as a `.pdf` file;
- a set of elevation maps.

2 The Steepest Descent Problem

2.1 General idea of the problem

Our data for this assignment consists in a 2D grid representing an elevation map (the altitude at each point on a grid).

Now, imagine that we can drop a skier at some random location on the grid. From that point, our valiant skier moves to the neighboring locating (using 8-connect[1], and once there moves again to the lowest neighbor, and keeps doing so until the skier's location is lower than that of all of its neighbors (in mathematical terms, we would say that the skier has reached a local minimum of the

[1] 8-connect is one of the two classical ways to define a neighborhood on a rectangular grid (the other being 4-connect). In our case, what it means is that a move is possible from a given square to any of its eight immediate neighbors (of course the number will be lower, 3 to 5 if the square lies on an edge of the map), allowing for diagonal displacements.

elevation function). At this point, the skier should report its final location and the elevation at that point and the path followed to reach that point. The central dispatcher could then arrange for the skier to be picked up and dropped to a new starting point, from which to repeat the procedure of going down to a new local minimum.

I am sure you see where this is headed: We are going to implement our dispatcher as a parent process and the skiers as its child processes, or possibly as threads within the skier process.

2.2 A few examples of trails

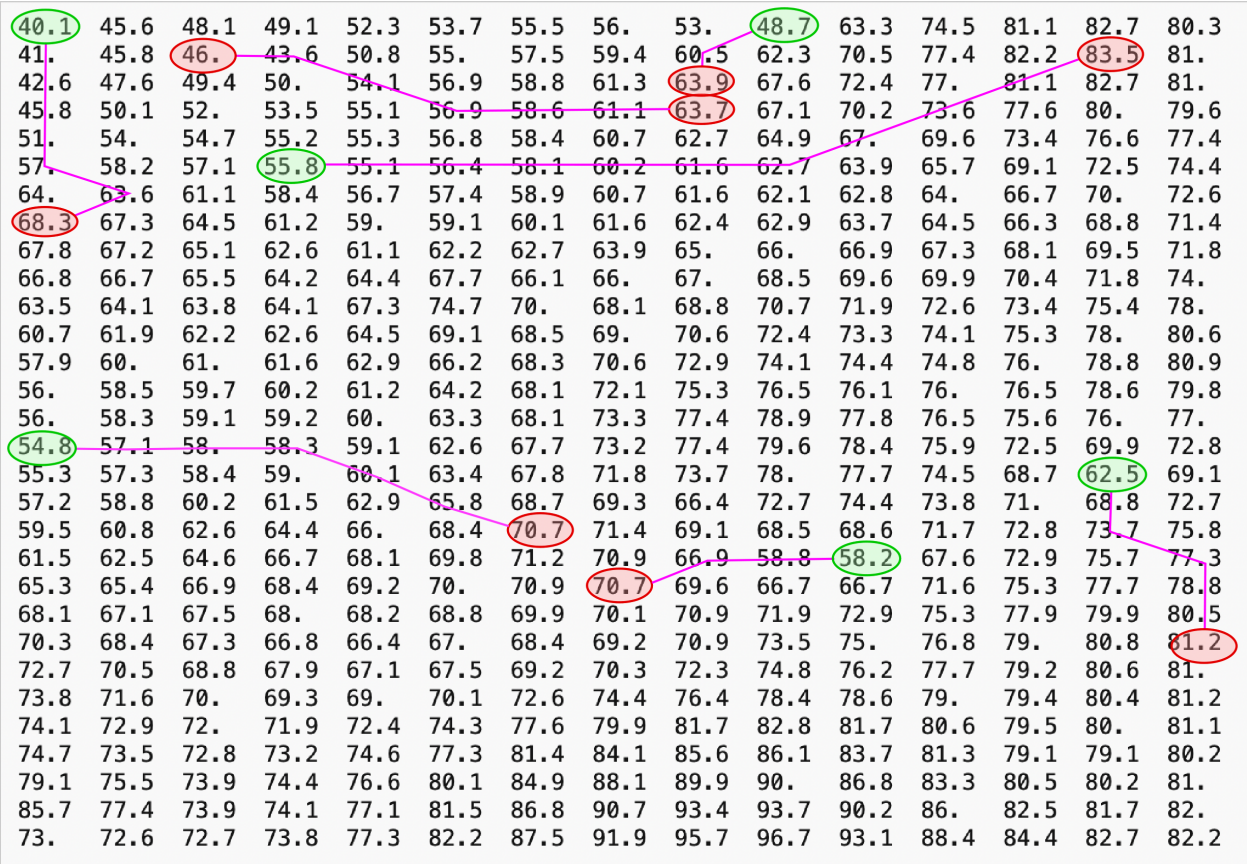

Figure 1: Examples of skier trails on the "map."

Figure 1 gives examples of skier trails obtained for different starting points on the map. Starting points are indicated by a red disk. The trails followed by different skiers are indicated as magenta-colored lines. All trails eventually end up at a point on the map that is lower than all the surrounding points, and there the skier's descent ends. Such endpoints are indicated by a green disk. As you can observe on the plot, two start points next to each other may give trails leading to very different endpoints.

2.3 Different sizes of maps

Figure 2 shows examples of elevation maps of different sizes displayed as gray-level images and as 3D surfaces. There is a family air between these different maps. It is an artifact of the algorithm that used to generate the maps (in case you care, it's a tweaked version of the Perlin noise generator, in which I had to force the presence of a few peaks and valleys for small-sized maps).

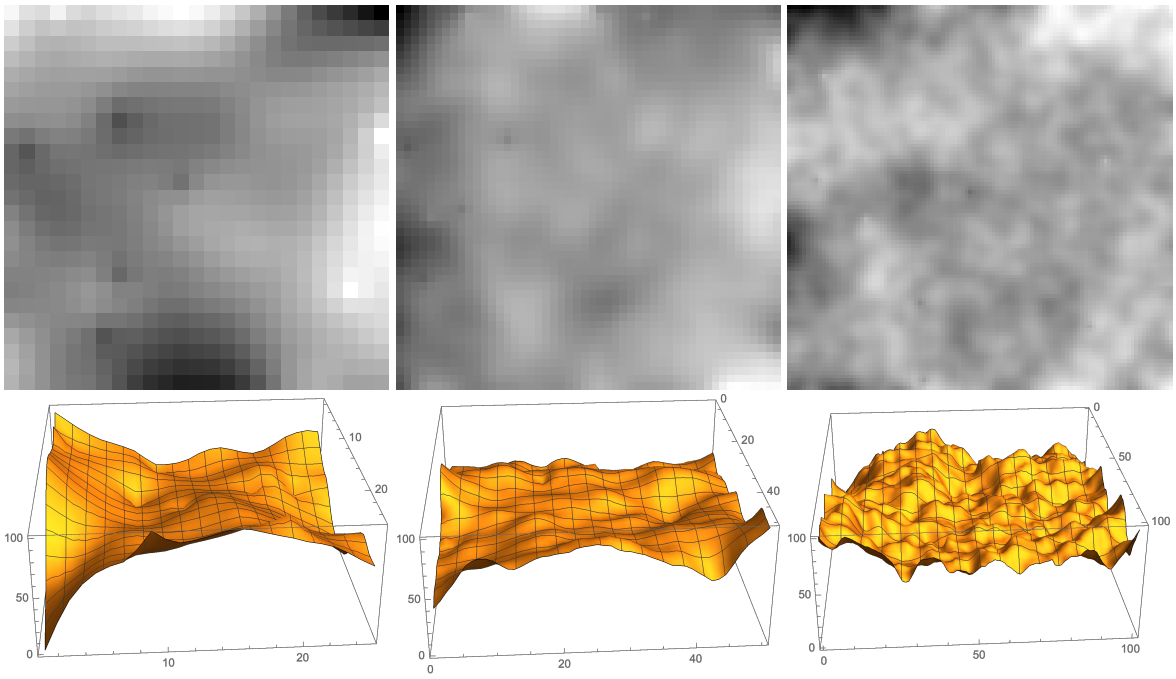

Figure 2: Examples of elevation maps of sizes 25×25, 50×50 and 100×100, respectively.

2.4 File format

The data files have the extension `.map`, but they are just plain text files. The file start with a line specifying the height and width of the map. the following lines (one per row of the map) list elevation values (a float number). Some maps are square (`_Sq`) suffix, others may be wider than they are high (`_Horiz` suffix) or higher than they are wide (`_Vert` suffix). As with all grid-based problem that don't absolutely require a square grid (e.g. games such as chess, checkers, or go), use mostly non-square grids so as to maximize your chances of exposing and spotting erroneous row-column order swap in your code.

I also post pdf file representing the map as gray-level images or surface plots, in case you want to check if the paths of your skiers make sense.

3 C++ Implementation

3.1 Dispatcher and skiers

Particularly when the map is large, reading the data file can take a significant amount of time. It is therefore something that we would rather not do more times than needed. This means that we should read the map in the parent process *before* the call to `fork()`, and then keep the data (which means not making an `execXY()` call, and therefore the parent "dispatcher" process and "skier" child processes will run off the same program).

The user (or script) will launch a dispatcher process, which in turn will create a number of "skier" child processes. After the skier processes have terminated, the dispatcher process should summarize their results and write them to a file.

3.2 Input to the parent process

As we shall see in the next section, the parent process will communicate with the bash script via a named pipe. The path to the named pipe will be part of the arguments passed to the parent process when it is launched. A minimal list of arguments would therefore contain

- the path to the elevation map data file;
- the path to an output folder where to write result files;
- the number of skier processes to create;
- the number of runs in the lifetime of a skier process;
- a flag `trace` / `notrace` indicating whether the skiers should run in *trace* mode.

3.2.1 Argument list

I specify the list of arguments that your program must take. This is an **ordered** list. It is complete. So, you cannot decide to take arguments in a different order and you cannot decide to omit some arguments, or to add arguments that I didn't require. If there is a problem with the list as requested in this document, discuss the issue with me, post a note on BrightSpace. If indeed I messed up, I will post a revision of the assignment, but don't make up your own personal "revised and improved" argument list. The grader can't deal with 40 different "personal" formats for the argument list.

3.3 Paths

Your programs and scripts must be able to handle any proper path. You cannot use hard-coded paths. You cannot assume that the path is relative to the local folder (i.e. the path doesn't necessarily start with `./`; it could very well be a full, absolute path starting from the root of the file system `/`). The path to a folder may end with a `/`,... or not.

3.4 Output of the parent process

The dispatcher process will print out to the output folder a file summarizing the runs of all the skiers. The output file will have the same base name as the map, but with the extension `.out` instead of `.map`.

The output file will have the following format:

- On the first line, the total number of runs cumulated by all the skiers, and the total number of different endpoints reached;
- Then for each endpoint reached, on a separate line, the row and column coordinates of the endpoint, followed by number of times it was reached;
- When in "trace" mode, a list of all the trace runs of all skiers, one run per line.

I don't specify the format of the output files of individual skiers. This is part of your design work (and this should be discussed in your report).

3.5 Multiprocessing

3.6 Version 1

In this version, each skier process is a single-threaded process that will perform the required number of runs of the map. For each run, a start position will be generated randomly.

3.7 Version 2

In this version, the separate runs will not be run sequentially, but as separate threads.

3.8 Trace mode

In either version, when running in "trace" mode, each skier process should output (to a file) the list of coordinates (in row-column order) of all the points the skier went through.

3.9 Extra credit 1 [3 points]

If there exists already an output file for the map, add an index to subsequent output files: `<map name>_2.out`, `<map name>_3.out`, etc.

3.10 Other extra credit

I may post an update version of this document with more EC options.

4 Scripting

4.1 Build script

The script named `build.sh`, when executed from the root `Prog05` folder, should build all the different versions of the C++ program that you have completed.

4.2 Watch folder

The idea for this part of the assignment is that your script should keep watch on a specific folder and process any file that the user drops in that folder:

- If the user dropped a map file, then the script should launch a "skier dispatcher" program to take care of that map;
- If the user dropped any other kind of file, then the script should simply ignore that file.

The arguments for this script, which must be named `script05.sh`, are, **in this specific order**):

1. The path to a "watch folder." If the folder doesn't already exist, your script should create it;
2. The path to the executable of a skier dispatcher program;
3. The number of skier processes to create;
4. The number of runs per skier;
5. A flag 0/1 indicating whether the skiers should run in "trace" mode;
6. The path to an output folder for the results of the skier processes.

Your script must be able to handle any proper path. You cannot use hard-coded paths; you cannot assume that the path is relative to the local folder (i.e. the path doesn't necessarily start with `./`; it could very well be a full, absolute path starting from the root of the file system `/`); the path may end with a `/`,... or not.

5 What to submit

5.1 The pieces

All your work should be organized inside a folder named `Prog05`. Inside this folder you should place:

- Your report;
- A folder named `Programs`, in which there should be a subfolder for all of the versions that you completed:

  - `Version_1` contains all the source and header files required to build the executable for Version 1 of the assignment;
  - `Version_2` contains all the source and header files required to build the executable for Version 2 of the assignment;

- A folder named `Scripts` where the `build.sh` and `script05.sh`;
- A folder named `Documentation` containing the html documentation produced by Doxygen (because, of course, you need to provide Doxygen documentation for this project).

Please note that we may test your program and script with data other than the ones that you provide, but at least we want to be able to test your code in the same conditions that you did.

5.2 What's in the report

The main sections I want to see in this report are:

- An explanation of the communications between script and parent process, and parent process and skier processes (how and when communication was set up, what get passed around, in what format, etc.);
- If you attempted to do any extra credit work, a list of what you did and how much of it was completed;
- A discussion of possible difficulties encountered in this assignment[2].
- A discussion of possible current limitations of your C/C++ program and script. Is it possible to make them either crash or fail?

5.3 Grading

- The C++ program follows the specifications (process creation, threading): 20 points
- The C++ program performs the task specified (execution): 30 points
- the `bash` script performs the task specified (execution): 15 points
- C and `bash` code quality (readability, indentation, identifiers, etc.): 15 points
- Report: 15 points
- Folder organization: 5 points

[2]Here, I don't mean "my laptop's battery is dying" or "I couldn't get a pizza delivered to my dorm room."