UNIX Shared Memory for Mandelbrot Colorization

Program Two
CS 3331 Spring 2023
Due: Feb. 15, Wednesday at 11:00pm

Motivation

UNIX allows independent processes to communicate through a shared region of memory. In this project, you will get experience with using shared memory. Additionally, you will get experience with the UNIX `fork /exec` paradigm.

The project is essentially two parts. First, you will convert your project one `pmandel` to use UNIX `fork` and `exec` instead of `fork` only. In the second part, you will implement a new color scheme. This scheme computes the color of each point from data that is computed and stored in shared memory by the child processes.

UNIX `fork /exec`

In the last project, you created the number of child processes specified on the command line. Each child process then executed a subroutine to complete is calculations. For this project, you must create a separate binary equivalent to the subroutine and then use `fork /exec` to run the child's calculations. Name this child file `mandelc.c`.

Note that the child behavior will be modified from project one. That is described below. You must create children that execute the new behavior specified below.

A New Color Scheme

For the second part of the project, a new color scheme will be implemented. The scheme is based on collecting, for each point evaluated, the number of iterations at which the recursion value exceeded 2, or the *escape* iterations. This is stored in the `pointCounts` array. (Remember that in the previous project, a color was assigned based on this iteration count and the color value was written to the PPM file.)

The `pointCounts` array is stored in shared memory. The parent creates shared memory to store the array and each child fills in the array for the strip that it is evaluating. When the children are done, the parent uses the `pointCounts` array to apply the new color scheme and creates the PPM file.

Figure 1 depicts the program operation.

Color Scheme Calculation

The new color scheme uses an `IterationMap`. The `IterationMap` is indexed by an iteration count, so the `IterationMap` array is sized to the maximum number of iterations specified on the command line. Each entry in the array maps an iteration count to the number of points for which the recursion exceeded two at that number of iterations. Note that the minimum number of iterations is 1. The will map to `IterationMap[0]`. You may create the IterationMap yourself, or use the code in `iterMap.c` available in the project directory.

The parent then calls the functions `iterSpectrumMap` and `spectrumToRGB`, which are available from the `/home/campus13/jmayo/public/cs3331/project2/` directory. These functions apply a color scheme using the `IterationMap` and `pointCounts` arrays and creates a PPM file.

A single-threaded program that performs these operations is given in `/home/campus13/jmayo/public/cs3331/project2/mandel-2.c`. Note that this is an example. It's expected that you would start with your project 1

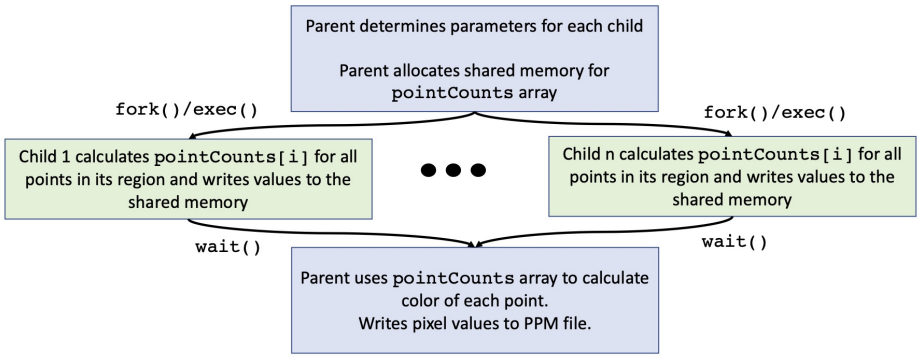

Figure 1: Process tree

code, not this code. You can use this code to understand the overall functionality and to check to make sure your application is performing correctly.

Following are two images you can use to determine if the correct images are being produced.

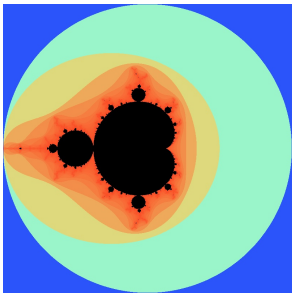

Figure 2: Parameters: -2 + 2I, 4.0 side length, 2000 iterations max, 800 pixels

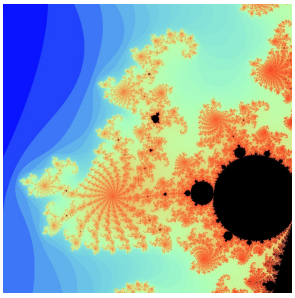

Figure 3: Parameters: -0.75 + 0.2I, 0.025 side length, 2000 iterations max, 800 pixels

Notes

- I suggest that you build the application up incrementally. Here is one approach.
    1. Implement the fork()/exec() model. It may be easier to retain the previous PPM file creation scheme so that you can validate the execution, but this PPM file construction will ultimately be stripped out.

2. Add communication through a shared memory segment. Make sure the child processes can write the segment and the parent can read data from all the children. If you are uncomfortable with the 2D array, start with something simpler.
3. Add the color scheme calculations.

- The main application will be implemented in a file named `pmandel-2.c` and will take the same parameters as for project one.
- Since the children do not write the PPM file, the `mandelc.c` binary will not take a file name argument. The remaining arguments should be (in order) (1) real coordinate of upper left corner, (2) imaginary coordinate of upper left corner, (3) side width on real axis, (4) side height on imaginary axis, (5) maximum number of iterations, (6) side width in pixels, and (7) side height in pixels. There can be no other parameters.
- The parent should delete the shared memory segment before it exits.
- Each child should output its PID and information about the strip it is working on as before.

Collaboration

Empty hands discussions are allowed for this project.

Submission

Keep all your files in a single directory to facilitate grading. Submit your files through Canvas. Your submission will contain `pmandel-2.c`, a `makefile`, `mandelc.c` and any other code or headers you use to complete the project. Typing `make` from a directory with your submitted files should create an executable named `pmandel-2` and an executable named `mandelc`. Typing `make clean` should remove all object files, binaries and image files.

Support for Grading

In order to support grading, implement the following targets in your makefile.

- `make dump` should build the system. It will be different from `make` in that it will compile `mandelc.c` with `DUMPPOINTS` defined (`gcc -DDUMPPOINTS`). When `DUMPPOINTS` is defined, each child should print in a legible fashion its PID and a dump of the `pointCounts` array that it calculated. Use `sprintf` and `write` to ensure the child output is not interleaved. Your `mandelc.c` file will then contain something like:

  ```
  #ifdef DUMPPOINTS
        write(1,pointprintbuf,strlen(pointprintbuf));
  #endif
  ```

  where `pointprintbuf` was created earlier. Be sure to show the indices of each `pointCounts` value shown. For example, your output might appear as:

  ```
  Row 3:  [0] 100   [1] 23   [2] 0 ...
  ```

  The default invocation of `make` should NOT define `DUMPPOINTS`. This make target allows us to check for correct values in the `pointCounts` array.

- `make watch` should build the system. It will be different from `make` in that it will cause the children to sleep for 8 seconds **immediately after their shared memory is attached**. In order to implement this functionality, compile `mandelc.c` to define `SLEEPINT`. When `SLEEPINT` is defined, it causes `sleep(8)` to be invoked. This make target allows us to check to see the shared memory structure is correct. The default invocation of make should NOT define `SLEEPINT`.

The project is due on Feb. 15, Wednesday, at 11pm. Be sure to check your submission after it is uploaded. Remember that submission of the wrong files will not be considered in the grading. Also be sure to run your code on `guardian.it.mtu.edu` or a CS lab machine. The fact that your code ran on your own machine will not be considered in the grading.