CPE-2600 FALL 2024

Systems Programming

Lab 11 Multiprocessing

Acknowledgement: Many aspects of this assignment are modeled after this: Operating Systems Principles (fiu.edu). See additional acknowledgements in the template code.

Topical Concepts

Multiprocessing

Although there are a number of benefits of multiprogramming, the most obvious is to achieve speedup of lengthy operations. To really explore these benefits, we have to have a lengthy operation. How about generating an image of a fractal? The Mandelbrot set is one such calculation that can be made and easily visualized. Depending on a variety of parameters, generation of a single image of the Mandelbrot set can take up to a few seconds on modern computing hardware. To aid in visualization often many images are calculated by varying parameters such as origin or scale factor then pieced into a movie. Examples can be seen here: Mandelbrot set - Wikipedia. The primary goal of this phase of the project will employ multiprocessing to enjoy a speedup while generating Mandelprot plots.

A program in the starter repository (mandel) has been provided that features a command line interface which allows adjustment of various parameters and will then generate a Mandelbrot set visualization and save it to a jpeg file.

Your assignment is to write a new program (mandelmovie) that will invoke mandel 50 times and vary parameters slightly with each new image to "fly" into the Mandelbrot set. This program, mandelmovie, will accept at least one command line argument which will be the number of processes used to calculate the frames. A value of 1 would mean 1 child process is used to invoke the mandel program. That child will generate the image then exit. The parent, upon detecting the child is finished will create a new child for the next frame and so on. A value of 2 would mean two child process would be created working on two images simultaneously, and so forth. Due to the nature of this setup, each child can operate independently once forked – no pipes are needed. Any supplied value up to the number of frames (at least 50) should function correctly. The parent process will need to wait until all children processes have finished before exiting.

You may either write a new program that uses fork/execl (or execv or related) to invoke the provided mandel program unchanged, or you may modify and refactor mandel to add this new feature.

Note that mandel uses 'getopt' to process command line arguments. You will need to use getopt for your new program or extend it to accommodate the new command line options if you choose to modify the existing program.

Once you have your 50 frames (or more if you wish) you can stitch them together into a movie using ffmpeg (you will need to install – sudo apt update / sudo apt install ffmpeg). Note, you will also need to install a development library to handle jpeg files to build mandel (sudo apt install libjpeg-dev).

Once everything is working, use the time command to measure the time it takes to generate the 50 images with various allowed numbers of children processes (1, 2, 5, 10, 20 for example). Plot the results. Does using more processes always speedup the operation? See below for the deliverable.

In Lab 12 we will investigate a multithreaded version.

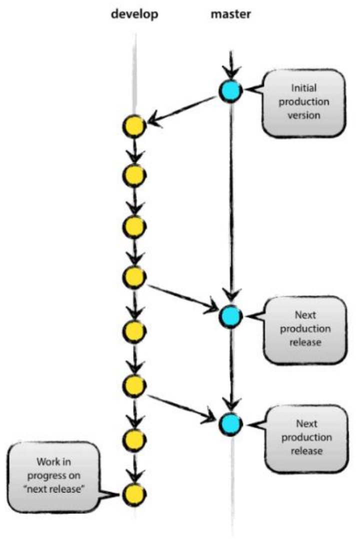
Git Source Version Control – Next Steps

Branches

So far we have been using git in its simplest form, really just as a tool to track a “linear” set of changes to a software project. We have an easy way to track changes, to mark milestones, and to undo changes. This is quite useful by itself if you are a solo developer and are not really sharing your code or repository with anyone.

In reality, you will rarely be a solo developer, and your repository will likely be accessed by other developers and stakeholders. So, we must introduce additional features to deal with more complicated scenarios. The first new concept is that of a branch. Like many things, this can get very complicated, so we will just be scratching the surface here as well.

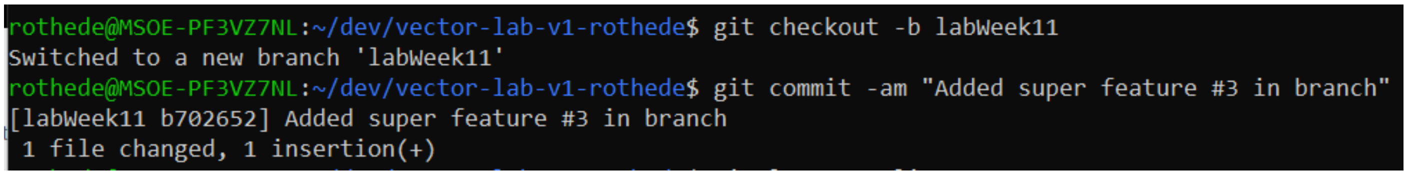
So far, we have just a single branch in our repository. By default, it is named “main.” As we made changes, we committed those changes to the main branch. However, if we are working with other developers or even just publishing our repository for public consumption of our project, there is a expectation that the main branch is always deployable. That is, a tested, stable version of the project. In addition, the numerous commits that you make leading up to that next stable version can ultimately create a lot of clutter.


develop
master
Initial production version
Next production release
Next production release
Work in progress on "next release"

So, you should be doing your actual development on a branch, leaving the main branch unchanged until the new features developed are fully tested and ready to deploy on the branch. At that point the branch can be merged back to the main branch. Some guidelines suggest maintaining a parallel development branch and only merge to main for releases.

Other guidelines suggest making a separate “feature” branches for a given set of features being added to the project. In this case the branch would have a descriptive name to reflect the features and not a generic name such as develop or dev.

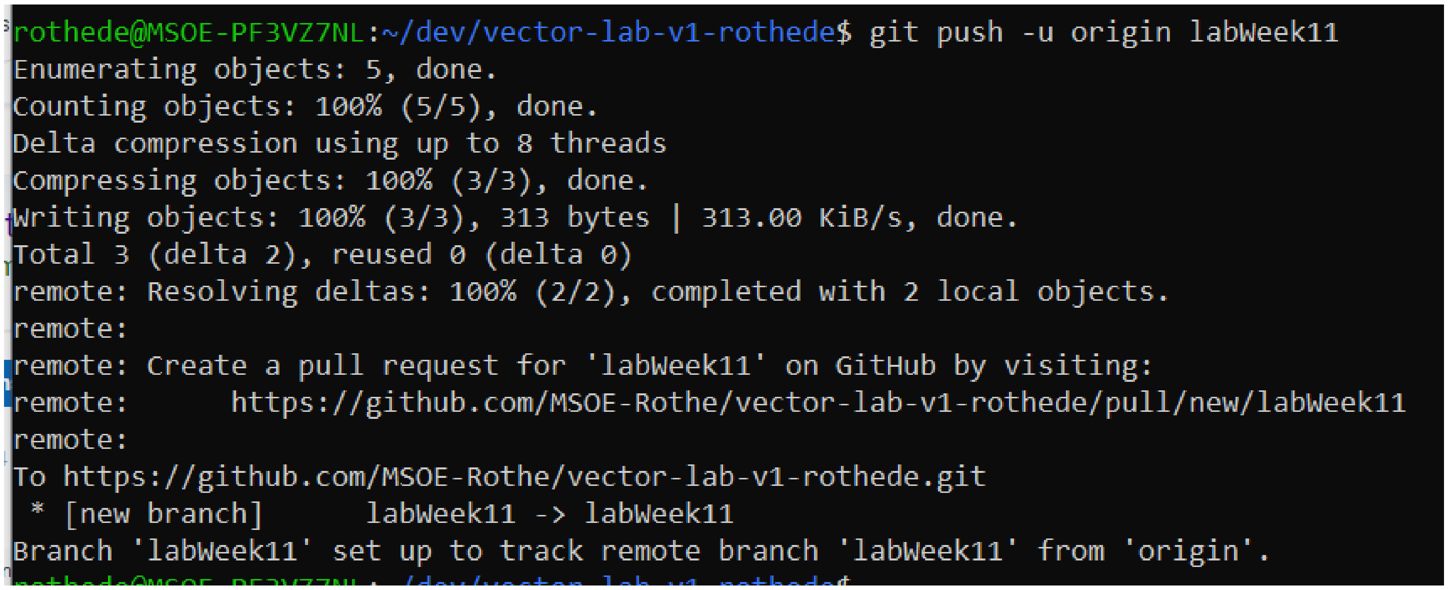
In any case, making the branch is easy. Assuming you are at the command line and currently at the HEAD of your main branch. You simply issue a ‘git checkout’ command like so:



```
rothede@MSOE-PF3VZ7NL:~/dev/vector-lab-v1-rothede$ git checkout -b labWeek11
Switched to a new branch 'labWeek11'
rothede@MSOE-PF3VZ7NL:~/dev/vector-lab-v1-rothede$ git commit -am "Added super feature #3 in branch"
[labWeek11 b702652] Added super feature #3 in branch
 1 file changed, 1 insertion(+)
```

With the -b option, this will create and switch to the new branch. Any commits made now will be on the branch. You can go back to the main branch with another git checkout command, but be careful. Any commits made on the main branch will complicate merging the branches later.

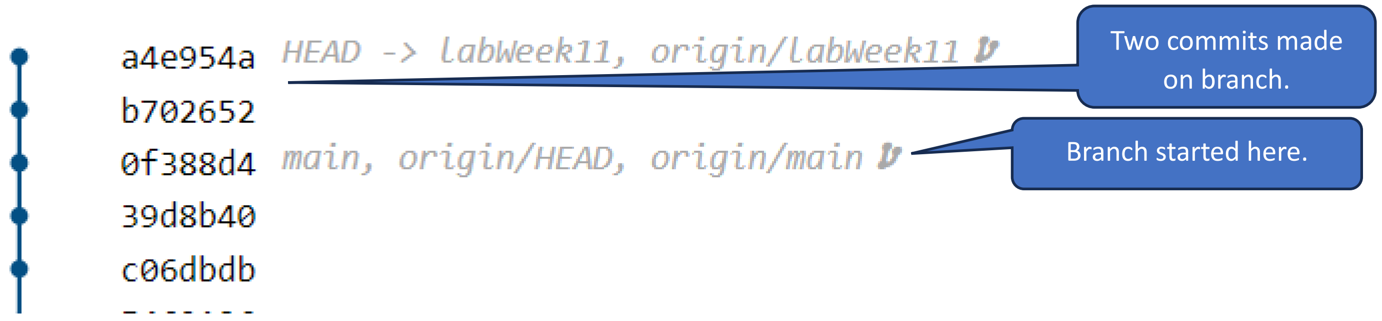
We can commit to this branch now, but, technically the branch does not exist in the remote repository, so right now, at least, we cannot push. To do that, there is one additional step we need to take. The following command will push to remote and specifically make the same branch on the remote repository.



```
rothede@MSOE-PF3VZ7NL:~/dev/vector-lab-v1-rothede$ git push -u origin labWeek11
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 313 bytes | 313.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'labWeek11' on GitHub by visiting:
remote:      https://github.com/MSOE-Rothe/vector-lab-v1-rothede/pull/new/labWeek11
remote:
To https://github.com/MSOE-Rothe/vector-lab-v1-rothede.git
 * [new branch]      labWeek11 -> labWeek11
Branch 'labWeek11' set up to track remote branch 'labWeek11' from 'origin'.
```

Now we can commit changes locally and push to remote the same way we have always done. Until we checkout a different branch or commit in the repo, we will be adding all commits on the branch (which is what we want).

Interestingly, as long as no commits are made to main, commits will still appear "linear." If you were to graph commits, you might see this:



```
a4e954a HEAD -> labWeek11, origin/labWeek11
b702652
0f388d4 main, origin/HEAD, origin/main
39d8b40
c06dbdb
```

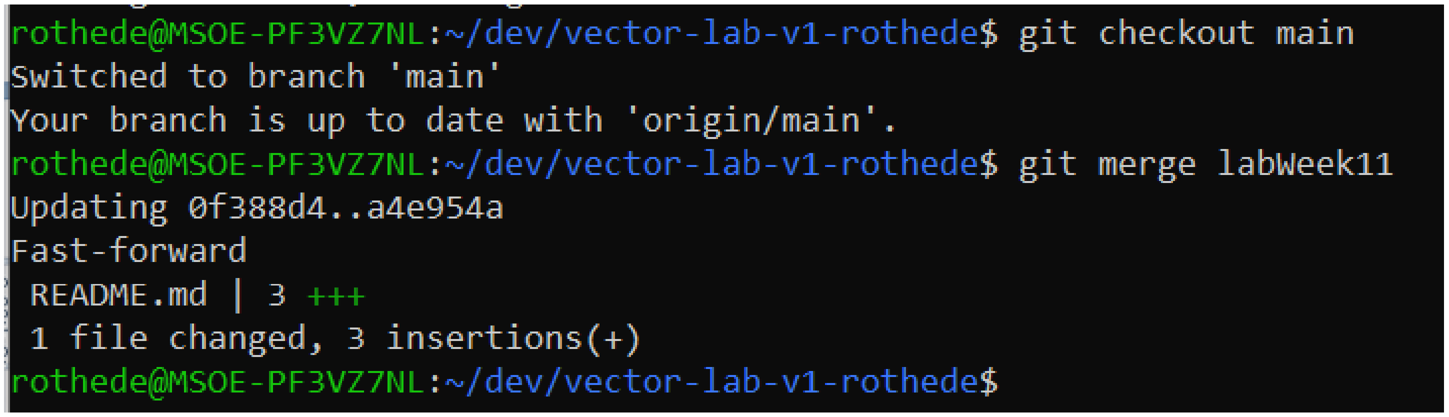
Merging

Ultimately, changes made on a branch will need to be merged back onto the main branch. As stated, ideally, no commits have been made to main since you are doing your work on a branch. So, the process of a merge is pretty trivial.

Be sure that you have no uncommitted changes when you start a merge, or you might wind up with some unintended issues.

If your current location (HEAD) is still on the branch, and you attempt to merge with main, nothing really happens because, although it is called a branch, if no commits have been made on main, there is no bifurcation in the history.

To successfully merge to main, you will need to change your location to the main branch (checkout) and issue the merge from there.


```
rothede@MSOE-PF3VZ7NL:~/dev/vector-lab-v1-rothede$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
rothede@MSOE-PF3VZ7NL:~/dev/vector-lab-v1-rothede$ git merge labWeek11
Updating 0f388d4..a4e954a
Fast-forward
 README.md | 3 +++
 1 file changed, 3 insertions(+)
rothede@MSOE-PF3VZ7NL:~/dev/vector-lab-v1-rothede$
```

Note the message – "Fast-forward". This basically acknowledges no changes were made to main since the branch. If changes had been made, it would be a "true merge" or a "three-way merge" which has a potential for conflicts. If there are conflicts you will have the opportunity to under the merge or proceed. If you proceed, your files will be marked where there are conflicts and you will need to manually fix them. A pain in a large project, for sure.

You can push to remote when done and have your new "release" ready to go.

The Exercise

Specific Instructions

1. Accept the new GitHub Classroom assignment at https://classroom.github.com/a/0nNkoC5K and clone the repository.

2. Before making any changes, create a branch name "labWeek11dev" and switch to that branch. Push the branch to the remote repo as shown above.

3. Build and become familiar with the mandel program. Experiment with various settings for the origin (-x and -y), the size of the image (-W and -H), scale (-s), etc. Identify a location you would be able to zoom in on for your movie. You might find this website (Mandelbots | Welcome to the Mandelbot Project) useful to guide your command line settings. You may also use $ ./mandel -h to see the command line options. You are welcome to modify any aspects you wish of the program to enhance the output. You may wish to tweak the function iteraction_to_color() in mandel.c to change color mappings.

4. As described above, either modify mandel or write a new program, mandelmovie, that will use child(ren) procresses to generate 50 images by progressively changing one or more image parameters (just changing scale will be easiest). The program must allow the number of children to be configured at the command line and the command line must be interpreted using the getopt function. *** **Enhancement; you may consider using a semaphore to manage the children processes.** ***

5. Collect runtime for 1, 2, 5, 10, and 20 children processes. Plot into a graph with # processes on the X axis and runtime on the Y axis. Prepare a brief report in README.md (edit the one in your repository). In the report, include:

   a. A brief overview of your implementation.
   b. The graph of your runtime results. You will need to export the plot (from Excel) into an image, add the image to your repo, and then link it into the README.md.
   c. A brief discussion of your results.

6. Create a movie of your 50 images. You can use the ffmpeg tool – this command should work: **`ffmpeg -i mandel%d.jpg mandel.mpg`**

   *** We will demo a few people's movie in the lab. Include your best movie in your repo for review and grading. ***

Deliverable

- When you are ready to submit your assignment prepare your repository:
- Make sure your name, assignment name, and section number are all files in your submission - in comment block of source file(s) and/or at the top of your report file(s).
- Make sure you have completed all activities and added all new program source files to repository.
- Make sure your assignment code is commented thoroughly.
- Make sure all files are committed and pushed to the main branch of your repository.
- Tag your repo with the tag ```vFinal```
- Include your best movie in the repo (we would not normally include such a file in a repo but for grading we do)

NOTE: Do not forget to 'add', 'commit', and 'push' all new files, branches, and changes to your repository before submitting.

To submit, copy the URL for your repository and submit the link to associated Canvas assignment and add a comment on Canvas that you have completed Lab 11.

The completed assignment is due as per Canvas date. You will be asked to demonstrate your program and/or play your movie.