Hints

- One way to get started is to modify `mr/worker.go`'s `Worker()` to send an RPC to the master asking for a task. Then modify the master to respond with the file name of an as-yet-unstarted map task. Then modify the worker to read that file and call the application Map function, as in `mrsequential.go`.
- The application Map and Reduce functions are loaded at run-time using the Go plugin package, from files whose names end in `.so`.
- If you change anything in the `mr/` directory, you will probably have to re-build any MapReduce plugins you use, with something like `go build -buildmode=plugin ../mrapps/wc.go`
- This lab relies on the workers sharing a file system. That's straightforward when all workers run on the same machine, but would require a global filesystem like GFS if the workers ran on different machines.
- A reasonable naming convention for intermediate files is `mr-X-Y`, where X is the Map task number, and Y is the reduce task number.
- The worker's map task code will need a way to store intermediate key/value pairs in files in a way that can be correctly read back during reduce tasks. One possibility is to use Go's `encoding/json` package. To write key/value pairs to a JSON file:

```
enc := json.NewEncoder(file)
for _, kv := ... {
  err := enc.Encode(&kv)
```

  and to read such a file back:

```
dec := json.NewDecoder(file)
for {
  var kv KeyValue
  if err := dec.Decode(&kv); err != nil {
    break
  }
  kva = append(kva, kv)
}
```

- The map part of your worker can use the `ihash(key)` function (in `worker.go`) to pick the reduce task for a given key.
- You can steal some code from `mrsequential.go` for reading Map input files, for sorting intermedate key/value pairs between the Map and Reduce, and for storing Reduce output in files.
- The master, as an RPC server, will be concurrent; don't forget to lock shared data.

- Use Go's race detector, with `go build -race` and `go run -race`. `test-mr.sh` has a comment that shows you how to enable the race detector for the tests.
- Workers will sometimes need to wait, e.g. reduces can't start until the last map has finished. One possibility is for workers to periodically ask the master for work, sleeping with `time.Sleep()` between each request. Another possibility is for the relevant RPC handler in the master to have a loop that waits, either with `time.Sleep()` or `sync.Cond`. Go runs the handler for each RPC in its own thread, so the fact that one handler is waiting won't prevent the master from processing other RPCs.
- The master can't reliably distinguish between crashed workers, workers that are alive but have stalled for some reason, and workers that are executing but too slowly to be useful. The best you can do is have the master wait for some amount of time, and then give up and re-issue the task to a different worker. For this lab, have the master wait for ten seconds; after that the master should assume the worker has died (of course, it might not have).
- To test crash recovery, you can use the `mrapps/crash.go` application plugin. It randomly exits in the Map and Reduce functions.
- To ensure that nobody observes partially written files in the presence of crashes, the MapReduce paper mentions the trick of using a temporary file and atomically renaming it once it is completely written. You can use `ioutil.TempFile` to create a temporary file and `os.Rename` to atomically rename it.
- `test-mr.sh` runs all the processes in the sub-directory `mr-tmp`, so if something goes wrong and you want to look at intermediate or output files, look there.

Handin procedure

> Before submitting, please run `test-mr.sh` one final time.

Use the `make lab1` command to package your lab assignment and upload it to the class's submission website, located at https://6824.scripts.mit.edu/2020/handin.py/.

You may use your MIT Certificate or request an API key via email to log in for the first time. Your API key (`XXX`) is displayed once you logged in, which can be used to upload lab1 from the console as follows.

```
$ cd ~/6.824
$ echo XXX > api.key
$ make lab1
```

Check the submission website to make sure it thinks you submitted this lab!

You may submit multiple times. We will use the timestamp of your **last** submission for the purpose of calculating late days.

Please post questions on Piazza.