This code is an implementation of a Coordinator for a MapReduce.

1. **Constants**:
    - `IDLE`, `IN_PROGRESS`, and `COMPLETED`: Constants representing different states of tasks.
2. **Structs**:
    - `MapTask`: Represents a map task with a filename and an index.
3. **Variables**:
    - `maptasks`: A channel for distributing map tasks.
    - `reducetasks`: A channel for distributing reduce tasks.
4. **Coordinator Struct**:
    - `mapTaskStatus`: Tracks the status of each map task.
    - `reduceTaskStatus`: Tracks the status of each reduce task.
    - `finish`: Indicates whether the entire job has finished.
    - `inputFiles`: List of input files for map tasks.
    - `nReduce`: Number of reduce tasks.
    - `mapIndex`, `reduceIndex`: Track current map and reduce indexes.
    - `intermediateFiles`: Stores intermediate files generated by map tasks.
    - `RWMutexLock`: A read-write mutex for synchronizing access to shared data.
    - `mapCompleted`, `reduceCompleted`: Flags indicating completion of map and reduce tasks.
5. **DistributeTask1 Method**:
    - Distributes either a map task or a reduce task to workers.
    - Monitors the completion of map tasks and reduce tasks.
6. **watchWorkerMap1 Method**:
    - Monitors the completion of a map task by a worker within a time limit.
    - Resets the task status to `IDLE` if not completed within the time limit.
7. **NotifyMapSuccess Method**:
    - Notifies the coordinator about the successful completion of a map task by a worker.
    - Checks if all map tasks are completed, then schedules reduce tasks.
8. **watchWorkerReduce Method**:
    - Monitors the completion of a reduce task by a worker within a time limit.
    - Resets the task status to `IDLE` if not completed within the time limit.
9. **NotifyReduceSuccess Method**:
    - Notifies the coordinator about the successful completion of a reduce task by a worker.
    - Checks if all reduce tasks are completed.
10. **NotifyIntermediateFile Method**:
    - Notifies the coordinator about intermediate files generated by map tasks.
11. **server Method**:
    - Starts an RPC server for handling communication with workers.

12. **Done Method**:
    - Checks if the entire MapReduce job is completed.
13. **MakeCoordinator Function**:
    - Initializes and returns a new Coordinator instance.
    - Initializes channels and maps, distributes initial map tasks, and starts the RPC server.

This code is a Worker implementation for the MapReduce .

1. **Constants and Variables**:
    - `ByKey`: Type definition for sorting `KeyValue` pairs by key.
2. **Structs**:
    - `KeyValue`: Represents a key-value pair, typically used in MapReduce tasks.
3. **Worker Function** (`Worker`):
    - Implements the worker's main functionality.
    - Continuously queries the coordinator for tasks (map or reduce).
    - Executes either a map task (`eMap`) or a reduce task (`eReduce`) based on the received task type.
    - Sends notifications to the coordinator upon completing map or reduce tasks.
4. **eMap Function**:
    - Executes a map task.
    - Reads content from the input file specified in the task.
    - Invokes the user-defined `mapf` function to process the input and generate key-value pairs.
    - Distributes the generated key-value pairs to appropriate reduce tasks.
    - Notifies the coordinator about the completion of the map task.
5. **eReduce Function**:
    - Executes a reduce task.
    - Reads intermediate files specified in the task, which contain key-value pairs.
    - Sorts the key-value pairs by key.
    - Invokes the user-defined `reducef` function to process the key-value pairs and generate the final output.
    - Writes the output to a file.
    - Notifies the coordinator about the completion of the reduce task.
6. **NotifyMapSuccess and NotifyReduceSuccess Functions**:
    - Send notifications to the coordinator about the successful completion of map and reduce tasks, respectively.
7. **ArrangeIntermediate Function**:
    - Groups key-value pairs by reduce task number (`NReduce`).
    - Used in the `eMap` function to distribute intermediate key-value pairs to reduce tasks.
8. **CallCoordinator Function**:
    - Makes an RPC call to the coordinator to get tasks (`DistributeTask1`).

    - Returns the received task and a boolean indicating whether the RPC call was successful.
9. **NotifyCoordinator Function**:
    - Notifies the coordinator about intermediate files generated during reduce tasks.
10. **call Function**:
    - Sends an RPC request to the coordinator and waits for a response.
    - Returns `true` if the RPC call was successful, `false` otherwise.

RPC (Remote Procedure Call) definitions for communication between the MapReduce coordinator and the worker nodes in RPC.go

1. **MrArgs struct**:
    - Represents the arguments for the main RPC calls used in the MapReduce system. It doesn't have any fields in this case.
2. **MrReply struct**:
    - Represents the reply for the main RPC calls. It contains fields to convey information related to MapReduce tasks such as map file name, task type (map or reduce), index, number of reducers (`NReduce`), and files (used in reduce tasks).
3. **NotifyIntermediateArgs struct**:
    - Represents the arguments for notifying the coordinator about intermediate files generated during reduce tasks. It contains fields for the reduce task index and the file name.
4. **NotifyMapSuccessArgs struct**:
    - Represents the arguments for notifying the coordinator about the successful completion of a map task. It contains the file name of the map task.
5. **NotifyReduceSuccessArgs struct**:
    - Represents the arguments for notifying the coordinator about the successful completion of a reduce task. It contains the reduce task index.
6. **NotifyReply struct**:
    - Represents the reply for the notification RPC calls. It doesn't have any fields in this case.
7. **coordinatorSock function**:
    - Generates the socket file name for communication between the coordinator and workers. It uses the user ID (`os.Getuid()`) to create a unique file name in the `/var/tmp` directory.