实验二、 MapReduce 的实现

一、流程及代码分析：

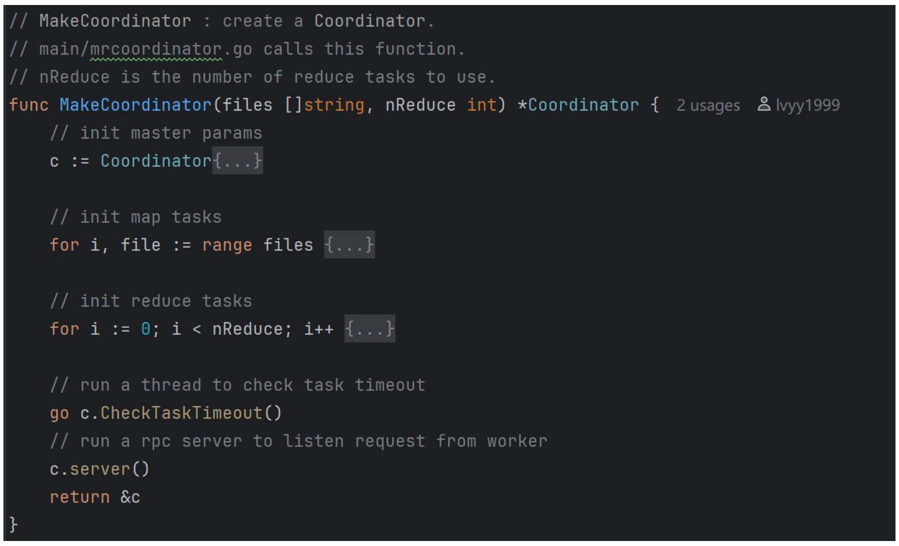
要求：通过流程图的形式，结合代码实现，分析 MapReduce 执行过程，并给出结果截图。

本次实验中的部分代码细节与原论文中的模型有所出入，例如：论文中的模型为 master 主动 ping 各个 worker 并分配任务，而在本次实验中采取的做法是由各个 worker 主动向 master 发起请求，master 将分配的任务信息放在 reply 中回复给 worker，这样做的好处是实现起来更为简便，只有 master 需要启动一个 rpc 服务进行监听，而 worker 作为 rpc 客户端调用即可。

master 流程分析：

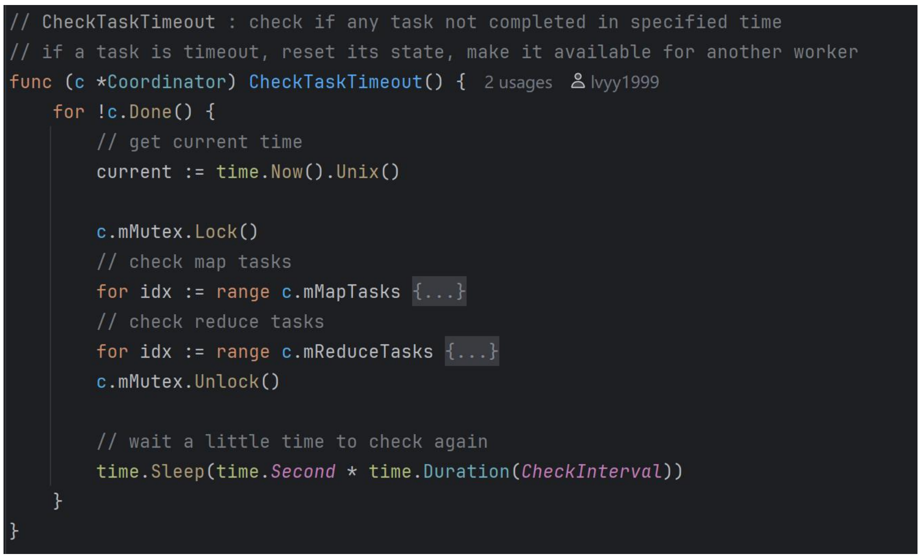
master（代码中叫 coordinator，二者是一个东西）只负责分配任务，由于 master 中并行执行了多个线程，就不对 master 画流程图了，直接用文字结合代码说明，后面的 worker 再画图：

1. 启动 master 并初始化参数，启动超时检查线程和 rpc 监听线程；



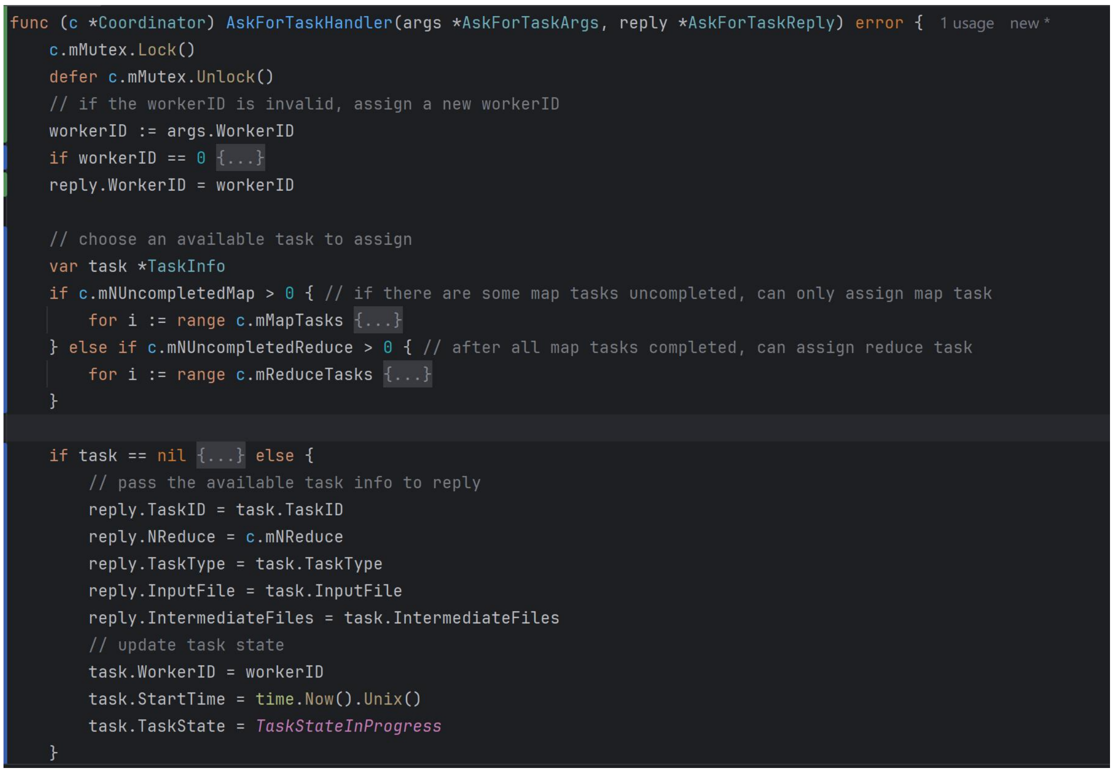
```go
// MakeCoordinator : create a Coordinator.
// main/mrcoordinator.go calls this function.
// nReduce is the number of reduce tasks to use.
func MakeCoordinator(files []string, nReduce int) *Coordinator {  2 usages  lvyy1999
    // init master params
    c := Coordinator{...}

    // init map tasks
    for i, file := range files {...}

    // init reduce tasks
    for i := 0; i < nReduce; i++ {...}

    // run a thread to check task timeout
    go c.CheckTaskTimeout()
    // run a rpc server to listen request from worker
    c.server()
    return &c
}
```

2. 定期检查是否有超时任务，若有，将其状态重置为空闲以便重新分配；



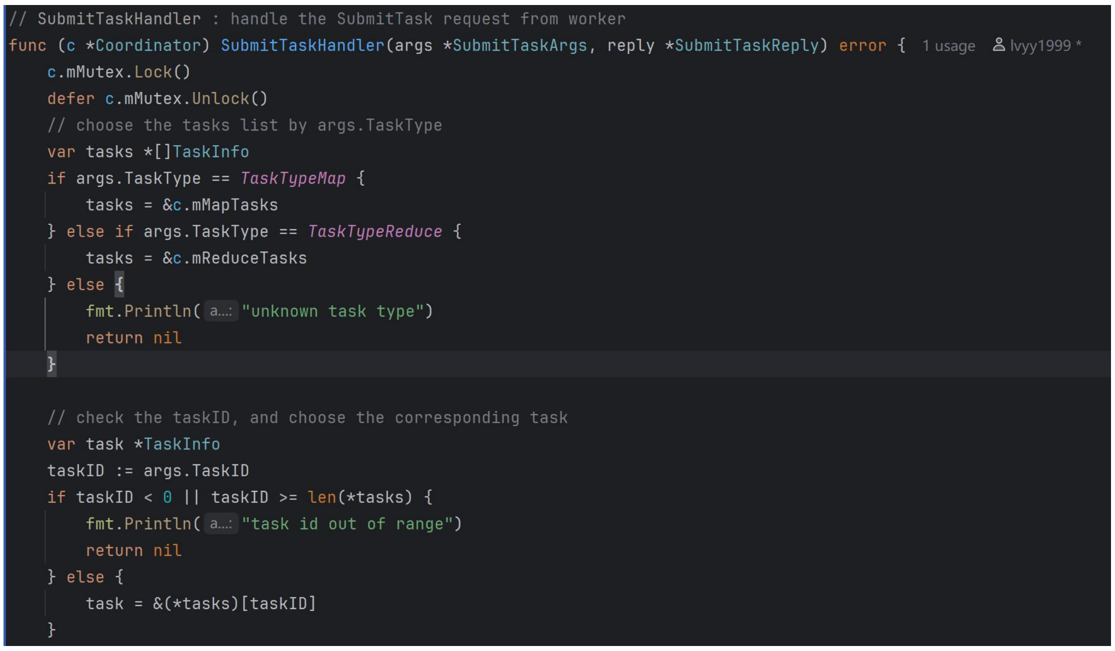
```go
// CheckTaskTimeout : check if any task not completed in specified time
// if a task is timeout, reset its state, make it available for another worker
func (c *Coordinator) CheckTaskTimeout() {  2 usages  lvyy1999
    for !c.Done() {
        // get current time
        current := time.Now().Unix()

        c.mMutex.Lock()
        // check map tasks
        for idx := range c.mMapTasks {...}
        // check reduce tasks
        for idx := range c.mReduceTasks {...}
        c.mMutex.Unlock()

        // wait a little time to check again
        time.Sleep(time.Second * time.Duration(CheckInterval))
    }
}
```

3. 当有 worker 来申请任务时，分配一个可执行的任务并更新任务状态。若有空闲的 map 任务则先分配 map 任务，只有当所有 map 任务都执行完才可以分配 reduce 任务，对于 reduce 任务，需要将 map 任务产生的中间文件地址告诉执行 reduce 任务的 worker；

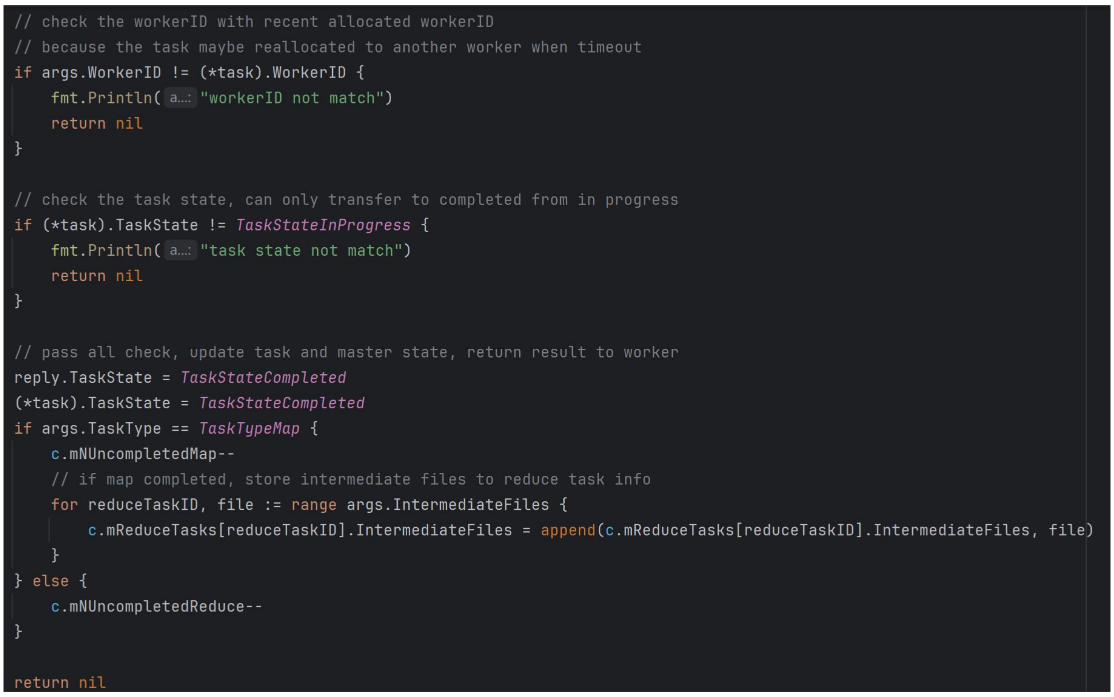


```go
func (c *Coordinator) AskForTaskHandler(args *AskForTaskArgs, reply *AskForTaskReply) error {  1 usage  new *
    c.mMutex.Lock()
    defer c.mMutex.Unlock()
    // if the workerID is invalid, assign a new workerID
    workerID := args.WorkerID
    if workerID == 0 {...}
    reply.WorkerID = workerID

    // choose an available task to assign
    var task *TaskInfo
    if c.mNUncompletedMap > 0 { // if there are some map tasks uncompleted, can only assign map task
        for i := range c.mMapTasks {...}
    } else if c.mNUncompletedReduce > 0 { // after all map tasks completed, can assign reduce task
        for i := range c.mReduceTasks {...}
    }

    if task == nil {...} else {
        // pass the available task info to reply
        reply.TaskID = task.TaskID
        reply.NReduce = c.mNReduce
        reply.TaskType = task.TaskType
        reply.InputFile = task.InputFile
        reply.IntermediateFiles = task.IntermediateFiles
        // update task state
        task.WorkerID = workerID
        task.StartTime = time.Now().Unix()
        task.TaskState = TaskStateInProgress
    }
```

4. 当有 worker 来提交任务时，检查信息是否匹配（如提交者的 id 和本地记录的执行者的 id 是否一致等），然后根据任务类型做相应处理（如若为 map 任务，需记录中间文件地址）；

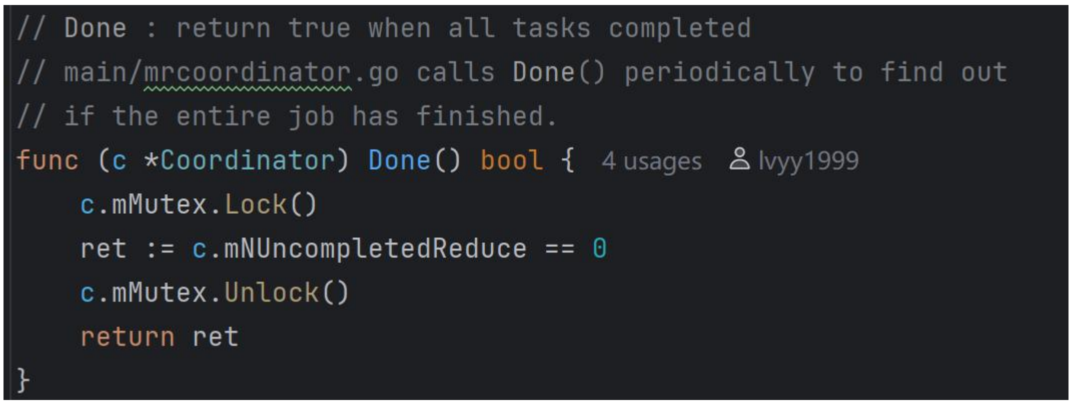


```go
// SubmitTaskHandler : handle the SubmitTask request from worker
func (c *Coordinator) SubmitTaskHandler(args *SubmitTaskArgs, reply *SubmitTaskReply) error {  1 usage  lvyy1999 *
    c.mMutex.Lock()
    defer c.mMutex.Unlock()
    // choose the tasks list by args.TaskType
    var tasks *[]TaskInfo
    if args.TaskType == TaskTypeMap {
        tasks = &c.mMapTasks
    } else if args.TaskType == TaskTypeReduce {
        tasks = &c.mReduceTasks
    } else {
        fmt.Println( a...: "unknown task type")
        return nil
    }

    // check the taskID, and choose the corresponding task
    var task *TaskInfo
    taskID := args.TaskID
    if taskID < 0 || taskID >= len(*tasks) {
        fmt.Println( a...: "task id out of range")
        return nil
    } else {
        task = &(*tasks)[taskID]
    }
```



```
// check the workerID with recent allocated workerID
// because the task maybe reallocated to another worker when timeout
if args.WorkerID != (*task).WorkerID {
    fmt.Println( a...: "workerID not match")
    return nil
}

// check the task state, can only transfer to completed from in progress
if (*task).TaskState != TaskStateInProgress {
    fmt.Println( a...: "task state not match")
    return nil
}

// pass all check, update task and master state, return result to worker
reply.TaskState = TaskStateCompleted
(*task).TaskState = TaskStateCompleted
if args.TaskType == TaskTypeMap {
    c.mNUncompletedMap--
    // if map completed, store intermediate files to reduce task info
    for reduceTaskID, file := range args.IntermediateFiles {
        c.mReduceTasks[reduceTaskID].IntermediateFiles = append(c.mReduceTasks[reduceTaskID].IntermediateFiles, file)
    }
} else {
    c.mNUncompletedReduce--
}

return nil
```

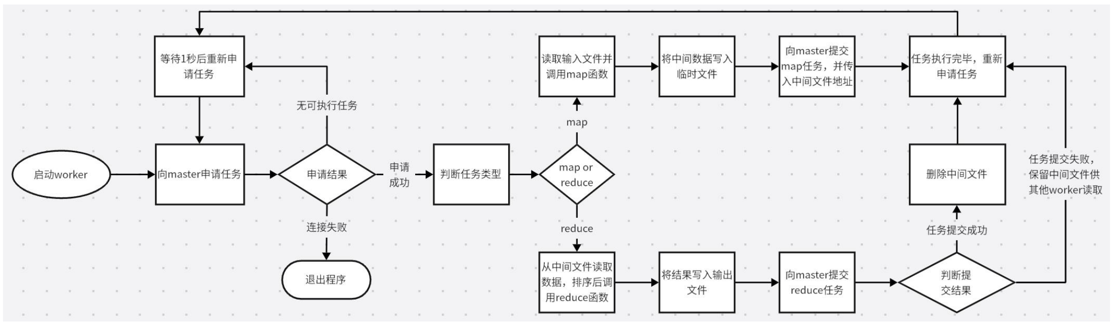
5. 当所有任务都执行完时，就可以退出了（这里不是直接退出，是模型调用者会不断调用 master 的 Done 接口查询工作完成情况，当返回 true 时调用者会退出），由于 map 任务都完成后才会分配 reduce 任务，所以这里判断 reduce 任务有没有全部完成就行了；



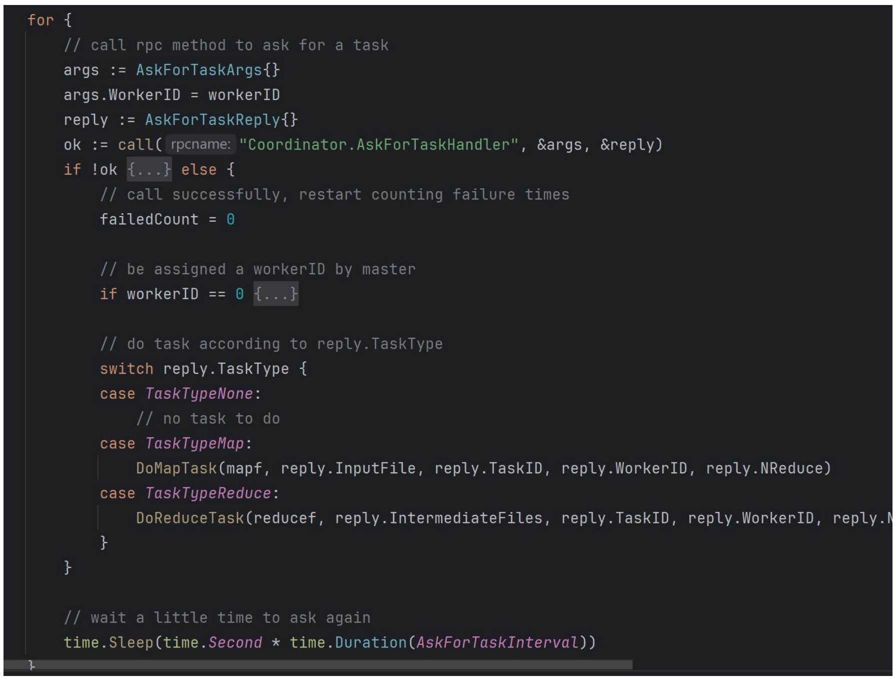
```
// Done : return true when all tasks completed
// main/mrcoordinator.go calls Done() periodically to find out
// if the entire job has finished.
func (c *Coordinator) Done() bool {  4 usages  lvyy1999
    c.mMutex.Lock()
    ret := c.mNUncompletedReduce == 0
    c.mMutex.Unlock()
    return ret
}
```

worker 流程分析：

worker 的流程相较 master 要复杂一些，主要流程图如下：

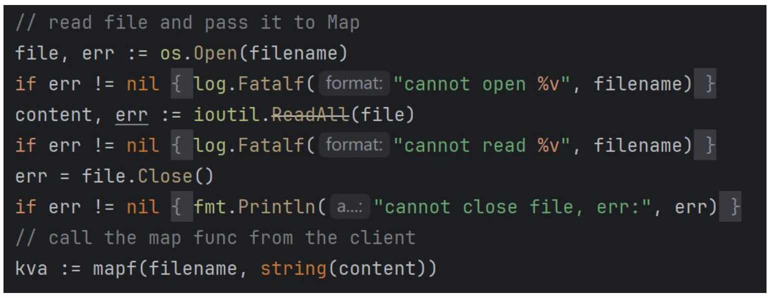


1. 启动 worker 并不断向 master 申请任务，根据任务类型转入相应处理函数；



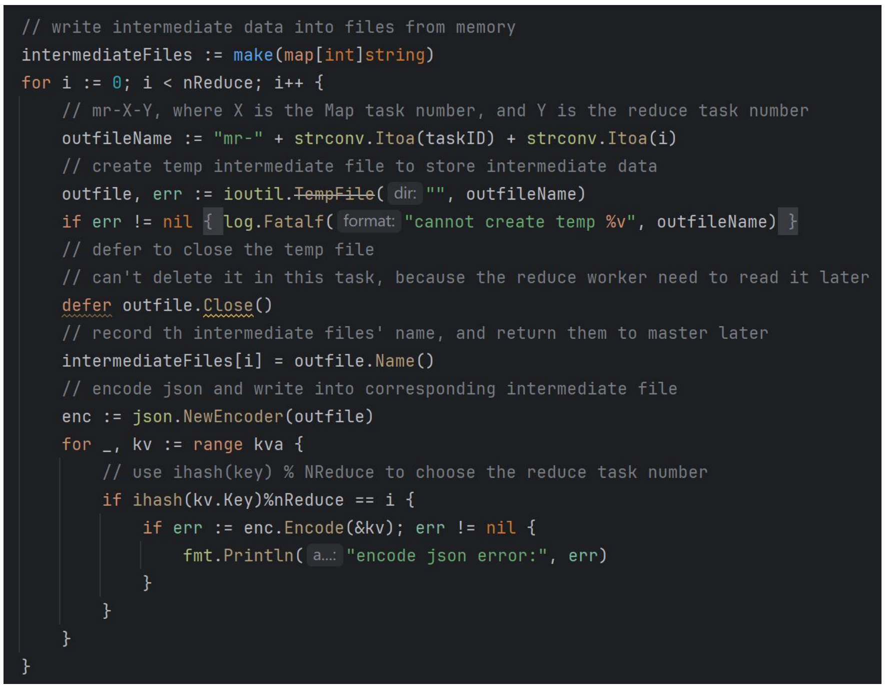
```go
for {
    // call rpc method to ask for a task
    args := AskForTaskArgs{}
    args.WorkerID = workerID
    reply := AskForTaskReply{}
    ok := call( rpcname: "Coordinator.AskForTaskHandler", &args, &reply)
    if !ok {...} else {
        // call successfully, restart counting failure times
        failedCount = 0

        // be assigned a workerID by master
        if workerID == 0 {...}

        // do task according to reply.TaskType
        switch reply.TaskType {
        case TaskTypeNone:
            // no task to do
        case TaskTypeMap:
            DoMapTask(mapf, reply.InputFile, reply.TaskID, reply.WorkerID, reply.NReduce)
        case TaskTypeReduce:
            DoReduceTask(reducef, reply.IntermediateFiles, reply.TaskID, reply.WorkerID, reply.N
        }
    }

    // wait a little time to ask again
    time.Sleep(time.Second * time.Duration(AskForTaskInterval))
}
```

2. 对于 map 任务：
2.1 读取输入文件并调用 client 提供的 map 方法；



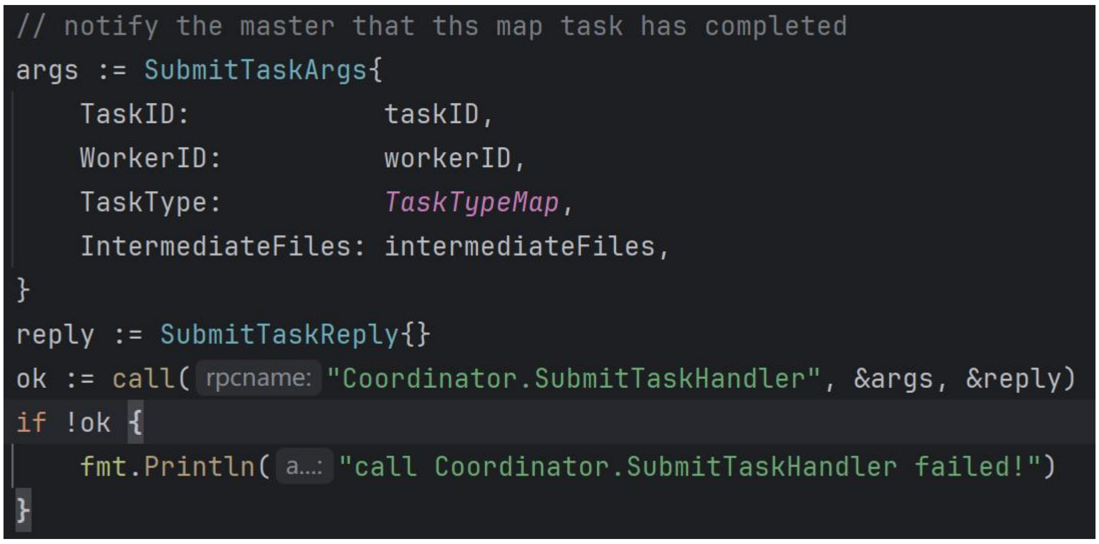
```go
// read file and pass it to Map
file, err := os.Open(filename)
if err != nil { log.Fatalf( format: "cannot open %v", filename) }
content, err := ioutil.~~ReadAll~~(file)
if err != nil { log.Fatalf( format: "cannot read %v", filename) }
err = file.Close()
if err != nil { fmt.Println( a...: "cannot close file, err:", err) }
// call the map func from the client
kva := mapf(filename, string(content))
```

2.2 将中间数据根据 key 进行 hash 分区，并以 json 格式写入相应的中间文件；



```go
// write intermediate data into files from memory
intermediateFiles := make(map[int]string)
for i := 0; i < nReduce; i++ {
    // mr-X-Y, where X is the Map task number, and Y is the reduce task number
    outfileName := "mr-" + strconv.Itoa(taskID) + strconv.Itoa(i)
    // create temp intermediate file to store intermediate data
    outfile, err := ioutil.~~TempFile~~( dir: "", outfileName)
    if err != nil { log.Fatalf( format: "cannot create temp %v", outfileName) }
    // defer to close the temp file
    // can't delete it in this task, because the reduce worker need to read it later
    defer outfile.Close()
    // record th intermediate files' name, and return them to master later
    intermediateFiles[i] = outfile.Name()
    // encode json and write into corresponding intermediate file
    enc := json.NewEncoder(outfile)
    for _, kv := range kva {
        // use ihash(key) % NReduce to choose the reduce task number
        if ihash(kv.Key)%nReduce == i {
            if err := enc.Encode(&kv); err != nil {
                fmt.Println( a...: "encode json error:", err)
            }
        }
    }
}
```

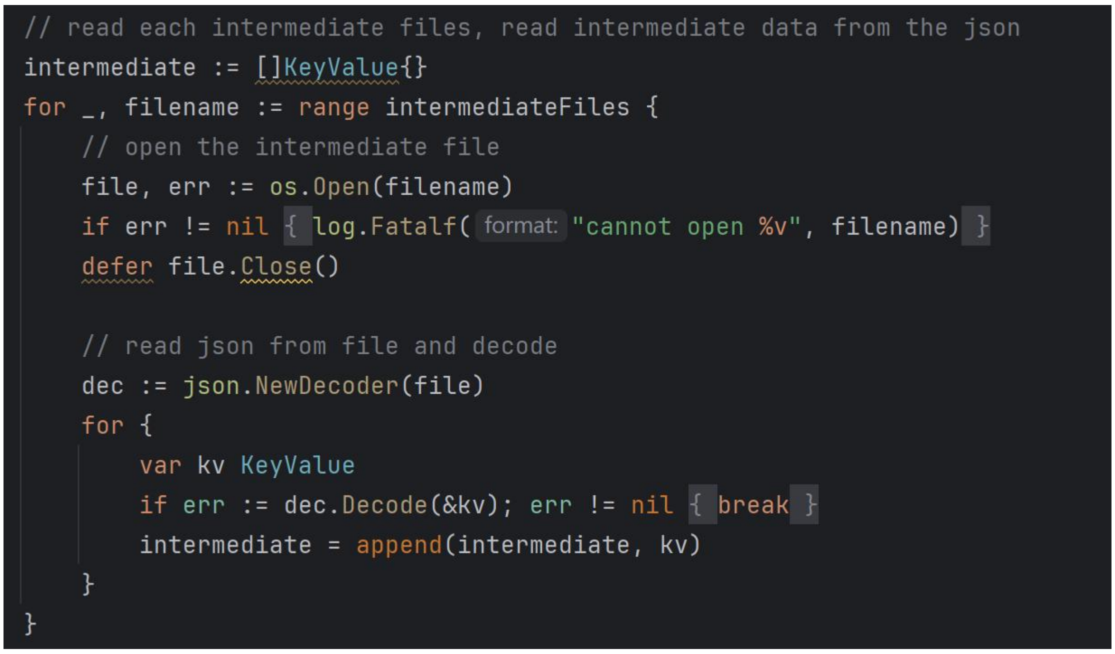
2.3 通知 master 任务执行完毕，并传入中间文件的路径地址；



```
// notify the master that ths map task has completed
args := SubmitTaskArgs{
    TaskID:            taskID,
    WorkerID:          workerID,
    TaskType:          TaskTypeMap,
    IntermediateFiles: intermediateFiles,
}
reply := SubmitTaskReply{}
ok := call( rpcname: "Coordinator.SubmitTaskHandler", &args, &reply)
if !ok {
    fmt.Println( a...: "call Coordinator.SubmitTaskHandler failed!")
}
```

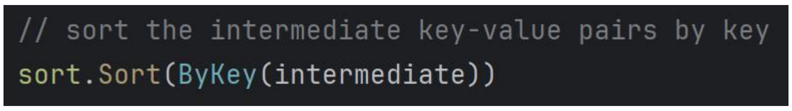
3. 对于 reduce 任务：

3.1 从 master 传入的中间文件地址读取 json 并解析成键值对；



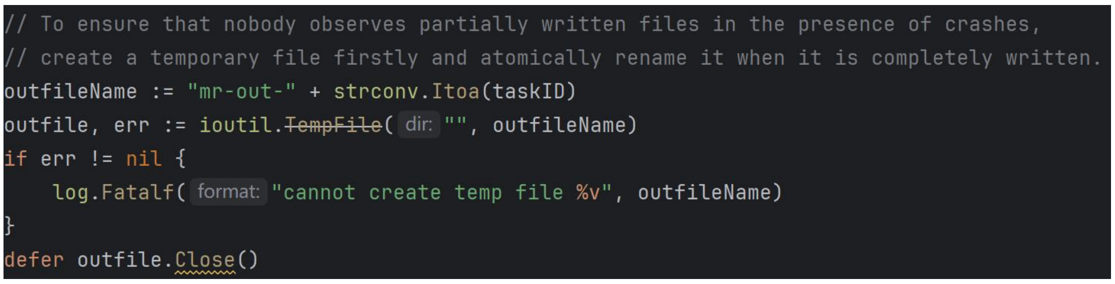
```
// read each intermediate files, read intermediate data from the json
intermediate := []KeyValue{}
for _, filename := range intermediateFiles {
    // open the intermediate file
    file, err := os.Open(filename)
    if err != nil { log.Fatalf( format: "cannot open %v", filename) }
    defer file.Close()

    // read json from file and decode
    dec := json.NewDecoder(file)
    for {
        var kv KeyValue
        if err := dec.Decode(&kv); err != nil { break }
        intermediate = append(intermediate, kv)
    }
}
```

3.2 对中间数据（键值对）进行排序；



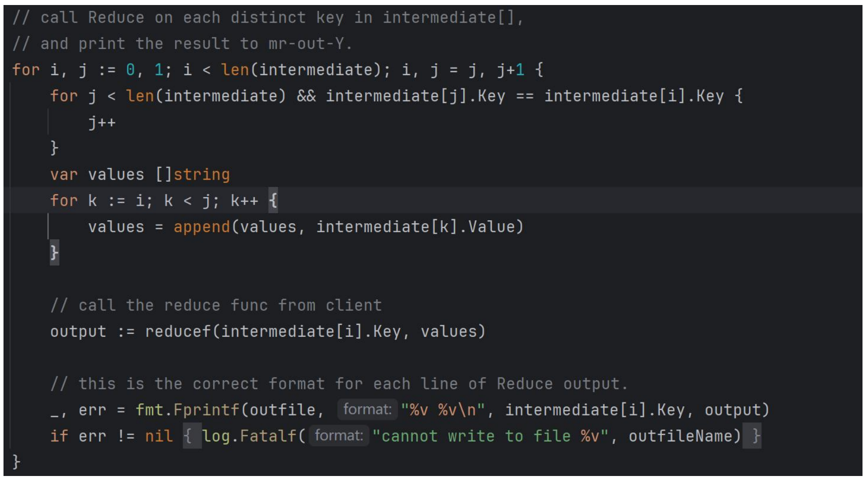
```
// sort the intermediate key-value pairs by key
sort.Sort(ByKey(intermediate))
```

3.3 先创建一个临时文件（防止文件在全部写入完成之前就被测试程序看到）；



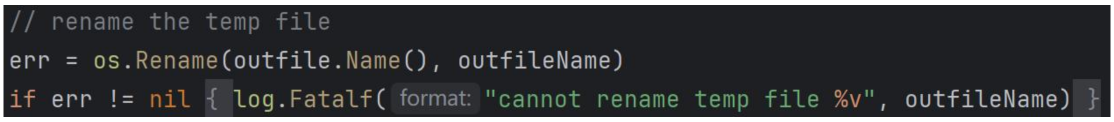
```
// To ensure that nobody observes partially written files in the presence of crashes,
// create a temporary file firstly and atomically rename it when it is completely written.
outfileName := "mr-out-" + strconv.Itoa(taskID)
outfile, err := ioutil.~~TempFile~~( dir: "", outfileName)
if err != nil {
    log.Fatalf( format: "cannot create temp file %v", outfileName)
}
defer outfile.Close()
```

3.4 将属于同一个 key 的 values 传入 client 提供的 reduce 函数，并将结果写入临时文件；



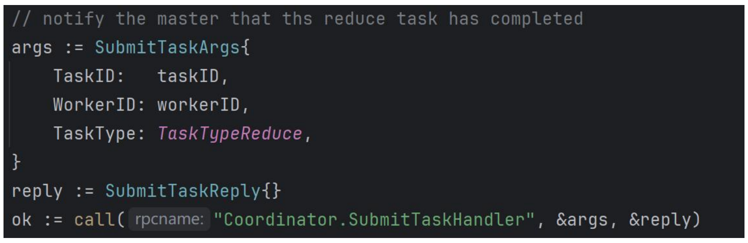
```go
// call Reduce on each distinct key in intermediate[],
// and print the result to mr-out-Y.
for i, j := 0, 1; i < len(intermediate); i, j = j, j+1 {
    for j < len(intermediate) && intermediate[j].Key == intermediate[i].Key {
        j++
    }
    var values []string
    for k := i; k < j; k++ {
        values = append(values, intermediate[k].Value)
    }

    // call the reduce func from client
    output := reducef(intermediate[i].Key, values)

    // this is the correct format for each line of Reduce output.
    _, err = fmt.Fprintf(outfile, format: "%v %v\n", intermediate[i].Key, output)
    if err != nil { log.Fatalf( format: "cannot write to file %v", outfileName) }
}
```

3.5 文件写好后，再通过 os.Rename 对其进行原子性地改名，使得其对测试脚本可见；



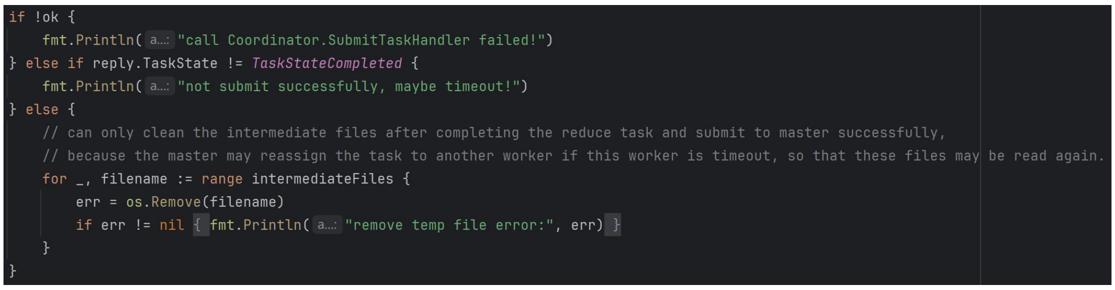
```go
// rename the temp file
err = os.Rename(outfile.Name(), outfileName)
if err != nil { log.Fatalf( format: "cannot rename temp file %v", outfileName) }
```

3.6 向 master 提交 reduce 任务已完成；



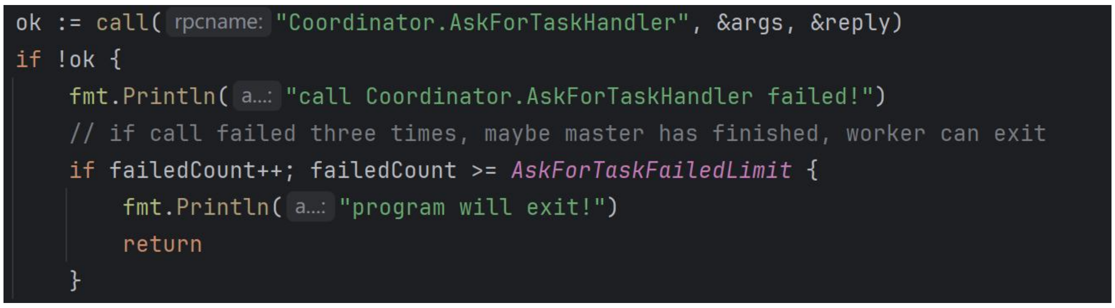
```go
// notify the master that ths reduce task has completed
args := SubmitTaskArgs{
    TaskID:   taskID,
    WorkerID: workerID,
    TaskType: TaskTypeReduce,
}
reply := SubmitTaskReply{}
ok := call( rpcname: "Coordinator.SubmitTaskHandler", &args, &reply)
```

3.7 若任务提交成功（调用 master 的任务提交接口成功且返回的任务状态为已完成），则可以删除 map 任务产生的那些存储中间数据的临时文件，以避免对存储的浪费；



```go
if !ok {
    fmt.Println( a...: "call Coordinator.SubmitTaskHandler failed!")
} else if reply.TaskState != TaskStateCompleted {
    fmt.Println( a...: "not submit successfully, maybe timeout!")
} else {
    // can only clean the intermediate files after completing the reduce task and submit to master successfully,
    // because the master may reassign the task to another worker if this worker is timeout, so that these files may be read again.
    for _, filename := range intermediateFiles {
        err = os.Remove(filename)
        if err != nil { fmt.Println( a...: "remove temp file error:", err) }
    }
}
```

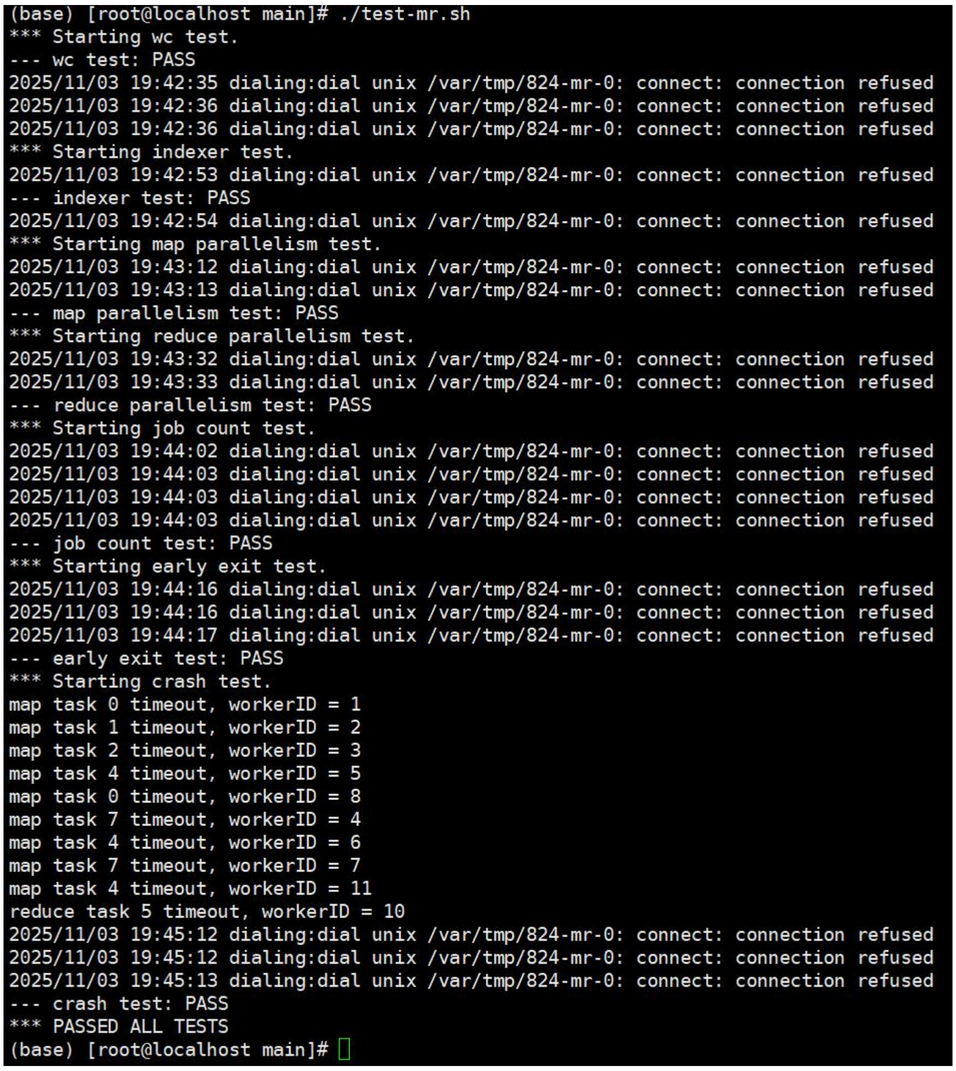
4. 若连接 master 失败，则可认为 master 已退出，那么 worker 也可以退出。



```go
ok := call( rpcname: "Coordinator.AskForTaskHandler", &args, &reply)
if !ok {
    fmt.Println( a...: "call Coordinator.AskForTaskHandler failed!")
    // if call failed three times, maybe master has finished, worker can exit
    if failedCount++; failedCount >= AskForTaskFailedLimit {
        fmt.Println( a...: "program will exit!")
        return
    }
```

二、实验结果截图：

本实验运行环境为 centOS7.4 虚拟机



```
(base) [root@localhost main]# ./test-mr.sh
*** Starting wc test.
--- wc test: PASS
2025/11/03 19:42:35 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:42:36 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:42:36 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
*** Starting indexer test.
2025/11/03 19:42:53 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
--- indexer test: PASS
2025/11/03 19:42:54 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
*** Starting map parallelism test.
2025/11/03 19:43:12 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:43:13 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
--- map parallelism test: PASS
*** Starting reduce parallelism test.
2025/11/03 19:43:32 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:43:33 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
--- reduce parallelism test: PASS
*** Starting job count test.
2025/11/03 19:44:02 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:44:03 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:44:03 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:44:03 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
--- job count test: PASS
*** Starting early exit test.
2025/11/03 19:44:16 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:44:16 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:44:17 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
--- early exit test: PASS
*** Starting crash test.
map task 0 timeout, workerID = 1
map task 1 timeout, workerID = 2
map task 2 timeout, workerID = 3
map task 4 timeout, workerID = 5
map task 0 timeout, workerID = 8
map task 7 timeout, workerID = 4
map task 4 timeout, workerID = 6
map task 7 timeout, workerID = 7
map task 4 timeout, workerID = 11
reduce task 5 timeout, workerID = 10
2025/11/03 19:45:12 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:45:12 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
2025/11/03 19:45:13 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused
--- crash test: PASS
*** PASSED ALL TESTS
(base) [root@localhost main]#
```

注：
1. 图片中的 dialing:dial unix /var/tmp/824-mr-0: connect: connection refused 报错是由于 worker 在连接 master 的 rpc 服务失败时才会退出，如果不想显示这个报错，可以修改 worker.go 中的 call 函数，去掉这个打印；或者由 master 在任务全部完成后，反向通知各个 worker 主动退出。
2. 项目提供的测试脚本中的 early exit test 一节，用到了 wait -n 命令，当系统的 bash 版本比较老时，会因为不支持这一命令导致这一项测试失败（没有等待任何进程退出就保存了结果，导致报错 output changed after first worker exited），需要将 bash 版本升级到 4.3 以上进行解决。

三、完整代码

由于代码较长，这里不贴出，可以直接看文件夹内的源代码文件，或从我的 github 仓库获取：https://github.com/lvyy1999/My6.824 。