Middle East Technical University | Department of Computer Engineering

CENG 334

Operating Systems

Spring 2018-2019

THE 1 - MapReduce

Due date: 17.03.2019, Sunday, 23:59

1 Objective

This homework aims to help you get familiar with,

- Forking and managing child processes
- Interprocess communication using pipes
- Duplicating file descriptors
- Reading from and writing to file descriptors
- Basics of MapReduce programming model

Keywords: *MapReduce, IPC, pipe, dup, exec*

2 Problem Definition

In this THE, you are going to implement a modified version of the MapReduce model. MapReduce was designed to process large data sets on clusters using parallel algorithms. It can be described as a system managing the data flow between each computing element of an algorithm. In other words, it manages communication between various nodes of a parallel algorithm designed to comply with the MapReduce programming model. MapReduce does not manage what an algorithm does, it manages how it does it.

A MapReduce program is composed of two main procedures, Map() and Reduce(). Each processing element executing Map() is called mapper and executing Reduce() is called reducer. Generally, MapReduce programs are executed with more than one mapper and reducer. In our case, each mapper and each reducer is executed as a separate process. Number of mappers and reducers in the system will always be equal to each other, which may not be the case for a real world MapReduce instance. In a MapReduce program, input data are initially divided into bunches based on a key K1 and each bunch is served to a separate mapper as input. Outputs generated by the mappers are combined and divided again using another key

K2. Then, each bunch is served to a reducer as input. Final outputs are generated by reducers for each key in K2. In our case, there is no separate key for Reduce(). Output of each mapper is forwarded directly to the reducer with the same id. Moreover, reducers forward their outputs to the next reducer instead of writing them to the output

3 Tasks

You have two tasks for this assignment. For the first task you will only implement the Map() procedure and for the second task you will implement both Map() and Reduce() procedures. Designing and implementing MapReduce algorithms is not in the scope of this assignment. You will only establish the communication paths. Example MapReduce algorithms will be provided later for you to test your implementation. Your program will be executed as:

```
./mapreduce N mapper_proc [reducer_proc]
```

where N is the number of mappers, mapper_proc is the program to be executed as mapper and reducer_proc is the program to be executed as reducer. When a mapper is executing its program it should give its ID to the program as argument. Same rule applies to the reducers. When your program has 3 arguments, it should establish Map model. When it has 4 arguments, it should establish MapReduce model. Note that reducer_proc argument will not be provided if Map model is to be established. Detailed explanation about both models can be found in Section 4. Your program should carry out following operations to establish the given model:

- Create necessary pipes (system call: pipe),
- Create child processes (system call: fork),
- Duplicate corresponding pipes to the specified file descriptors of the child processes and close unused end of pipes (system call: dup2, close),
- Execute mappers and reducers (if necessary) (system call: exec family),
- Feed mappers with the given inputs (system calls: read, write),
- Wait until all child processes terminate and then terminate itself (system call: wait family).

Closing unused end of pipes is critical. For EOF (end of file) to be sent to a pipe, its write end should not be open in any active process. Your program should close all unused pipes in parent and all child processes after they are forked. Duplicated file descriptors should also be closed after there is nothing to write to or read from them. Otherwise, the system may not give any result and lock indefinitely.

4 Implementation Details

You are going to implement two related models in this assignment, Map and MapReduce. The model to be established with 3 command line arguments is the Map model and the the model to be established with 4 command line arguments is the MapReduce model. You can see the general overview of both models from figures 1 and 2.

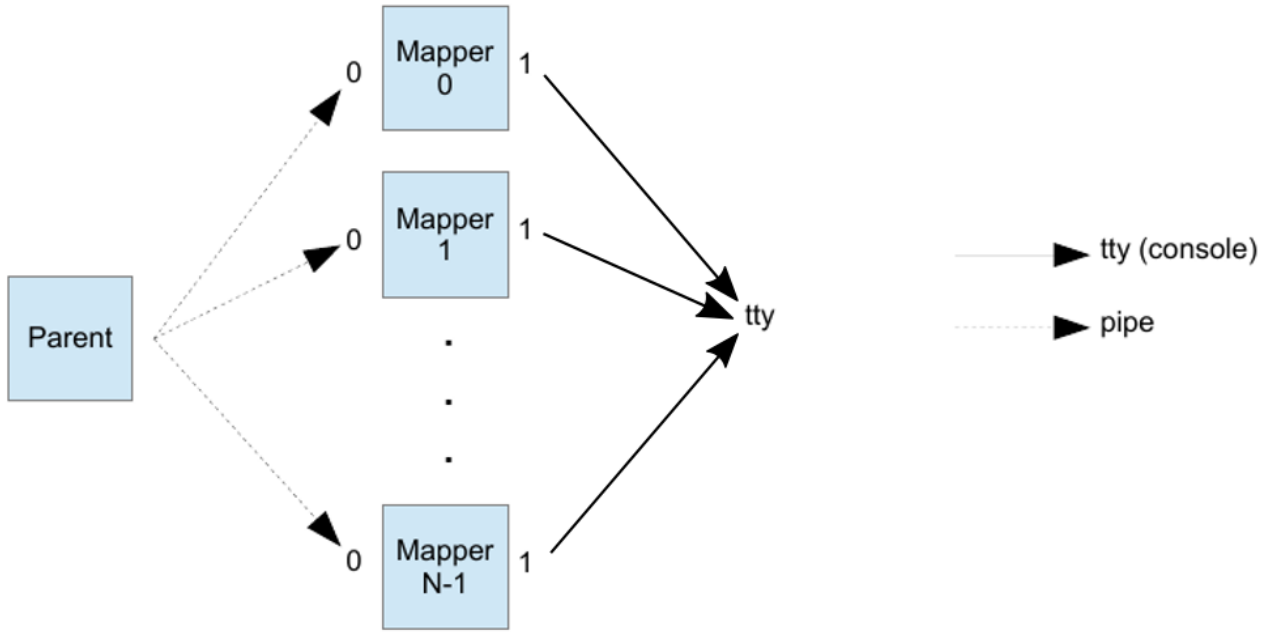

Figure 1: Overview of the Map model.

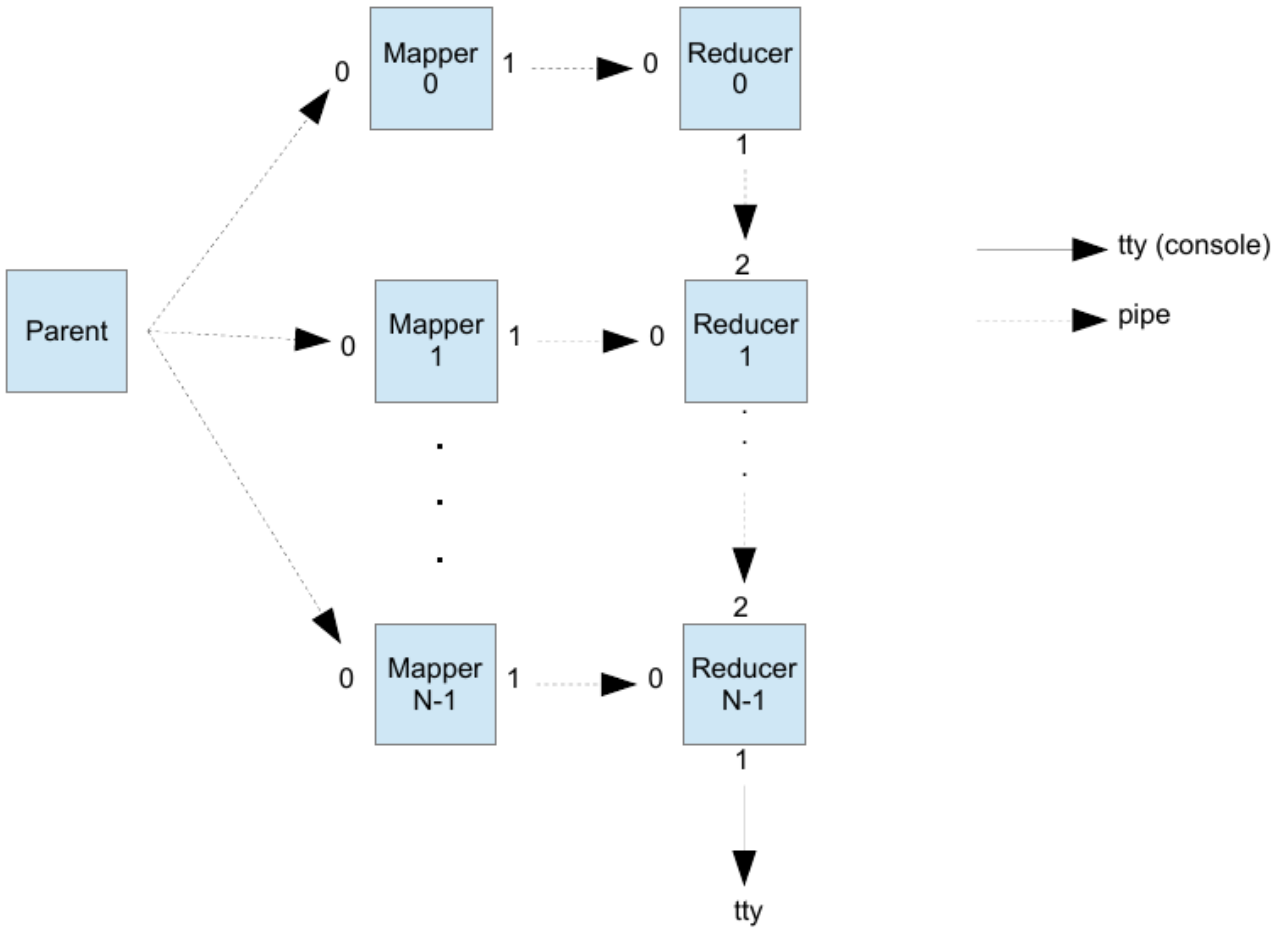

Figure 2: Overview of the MapReduce model.

Your program will be the parent in both of these models. Its job is to create the necessary child processes and pipes in the models. Since pipes are unidirectional communication channels, you should close the read end of the origin and write end of the destination for all the pipes. When executing the map and reduce programs, you should give the ID of the executing mapper or reducer to the program as argument.

In the Map model, your program should duplicate the pipes coming from the parent to standard input of the mappers. At this stage standard output of the mappers should not be duplicated, as we want the output to be written on the console. The parent should read the input from its standard input and distribute it among the mappers. The input is separated by lines. The i^{th} line of the input should be sent to the mapper with ID, $(i \ mod \ N) - 1$, where N is the number of mappers. For example, assume that $N = 3$. The first 3 lines of the input is sent to mappers with ID's 0,1 and 2, respectively. After that, the next line is sent to Mapper 0, the line after that is sent to Mapper 1 and so on. Newline characters at the end of the lines should also be sent. Parent process should continue reading and distributing the input until it reads EOF from stdin. In that case it should close all the pipes to the mappers in order to send EOF to the mappers. The parent process should wait all the mappers to terminate before terminating itself.

In the MapReduce model, the parent process also should set up communications related to the reducers. In this model there is a pipe between Mapper i and Reducer i for all i which is duplicated to the stdout of Mapper i and stdin of Reducer i. Also there is a pipe between Reducer i and Reducer $i+1$ for all i except $i = N-1$, which is duplicated to stdout of Reducer i and stderr of Reducer $i+1$. stdout of Reducer $N-1$ should not be changed since it should write to stdout.

5 Regulations

- **Programming Language:** You must code your program in C. Your submission will be compiled with `gcc` on department lab machines. You are expected make sure your code compiles successfully with `gcc` on depertment lab machines.
- **Input:** The input will be provided from stdin and consist of ASCII characters only. EOF will be sent at the end of the input.
- **Late Submission:** For every late day, $5 * day * day$ penalty will be applied.
- **Cheating:** Using any piece of code that is not your own is strictly forbidden and constitutes as cheating. This includes code from friends, previous homeworks, or the internet. The violators will be punished according the department regulations.
- **Discussion:** Follow the course page on Piazza for any updates and clarifications. Please ask your questions on Piazza instead of e-mailing if the question does not contain any code or solution.
- **Grading:** This homework will be graded out of 100. It will make up 10% of your total grade.
- **Evaluation:** Your program will be evaluating using black-box technique. You should ensure that you program is executing correctly when its stdin and stdout is directed to a file. Ex:

```
./mapreduce 10 countwords_map countwords_reduce > output.txt < input.txt
```

6 Submission

Submission will be done via COW. You will submit a tar file called "the1.tar.gz" that contain all your source code together with your Makefile. Your tar files should not contain any folders. Your Makefile should be able to create an executable named mapreduce and compile using the following commands.

```
> tar -xf the1.tar.gz
> make all
```