

[System Programming] Assignment #3

Spring 2025

Seulki Lee
Yunseok Lee
Kyu Hwan Lee
Jiwoon Chang
Yejin Lee

CONTACT

Ulsan National Institute of Science and Technology

Address 50 UNIST-gil, Ulsu-gun, Ulsan, 44919, Korea

Tel. +82 52 217 0114 **Web.** www.unist.ac.kr

Computer Science and Engineering

106 3rd Engineering Bldg

Tel. +82 52 217 6333 **Web.** <https://cse.unist.ac.kr/>

Assignment #3 (100 points)

- Due May 9, 2025: 11:59pm
- Platform
 - We will work on Ubuntu 22.10 (latest version)
 - <https://releases.ubuntu.com/kinetic/> (Desktop image)
- If you use MAC, please use Docker Desktop on Mac
 - <https://docs.docker.com/desktop/install/mac-install/>
- If you cannot make this environment, please contact our TA
 - Yunseok Lee: walk1009@unist.ac.kr
 - Kyu Hwan Lee: hanbitchan@unist.ac.kr
 - Jiwoon Chang: jwc9876@unist.ac.kr
 - Yeojin Lee: yeojin@unist.ac.kr

A basic multi-process program

- This assignment is to implement a basic multi-process program.
- Your program MUST take a path of a text file and the number of processes as arguments.
 - Thus, you should run your program like
`./assignment3 input.txt 4`
- Your program MUST read a text file.
 - The text file contains a simple string on each line and has multiple lines more than the number of processes
- Your program MUST read the file by using 4 to 16 processes as follows
 - The following example uses 5 processes, but your code MUST use 4 to 16 processes depending on the argument input values

PIDs

1

Read a line of the text file
Print the line with its PID
Let the next process read a line

2

Read a line of the text file
Print the line with its PID
Let the next process read a line

3

Read a line of the text file
Print the line with its PID
Let the next process read a line

4

Read a line of the text file
Print the line with its PID
Let the next process read a line

5

Read a line of the text file
Print the line with its PID
Let the next process read a line

Text file

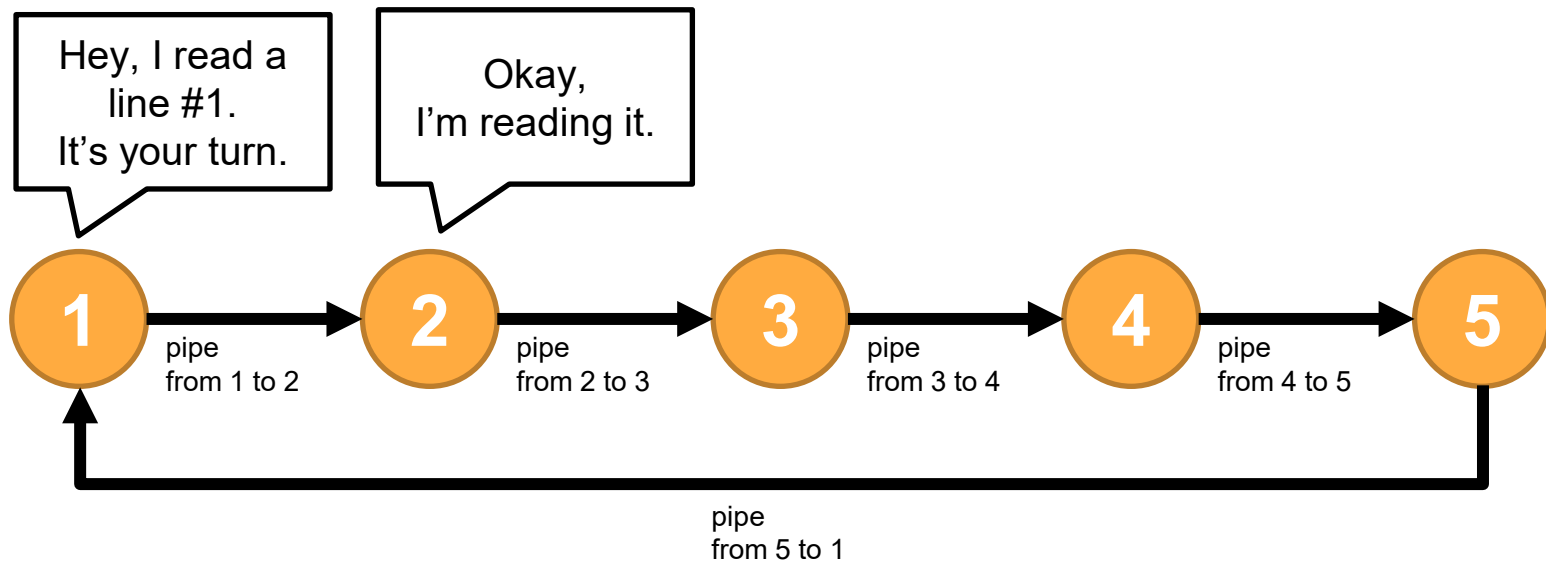
1 first
2 second
3 third
4 fourth
5 fifth
1 sixth
2 seventh
3 eighth
4 ninth
5 tenth
1 eleventh
2 twelveth
3 thirteenth
4 fourteenth
5 fifteenth

...

...

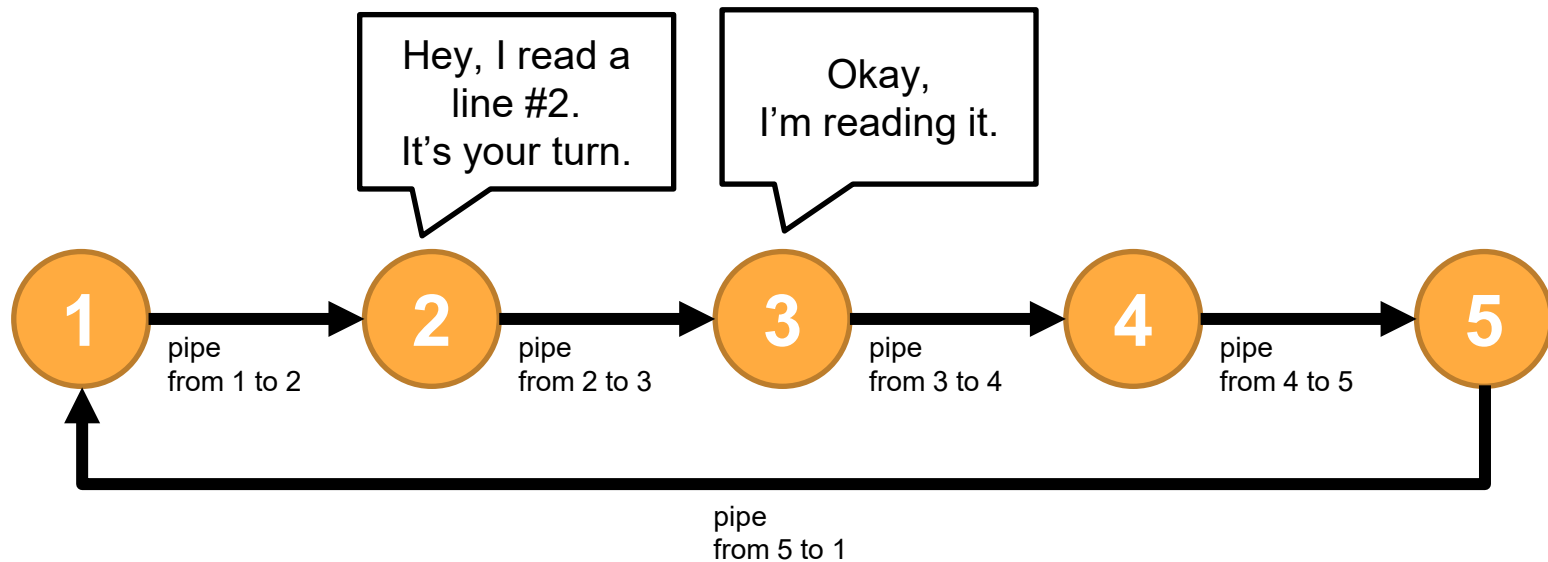
How can we let the next process read a line?

- PIPE!!



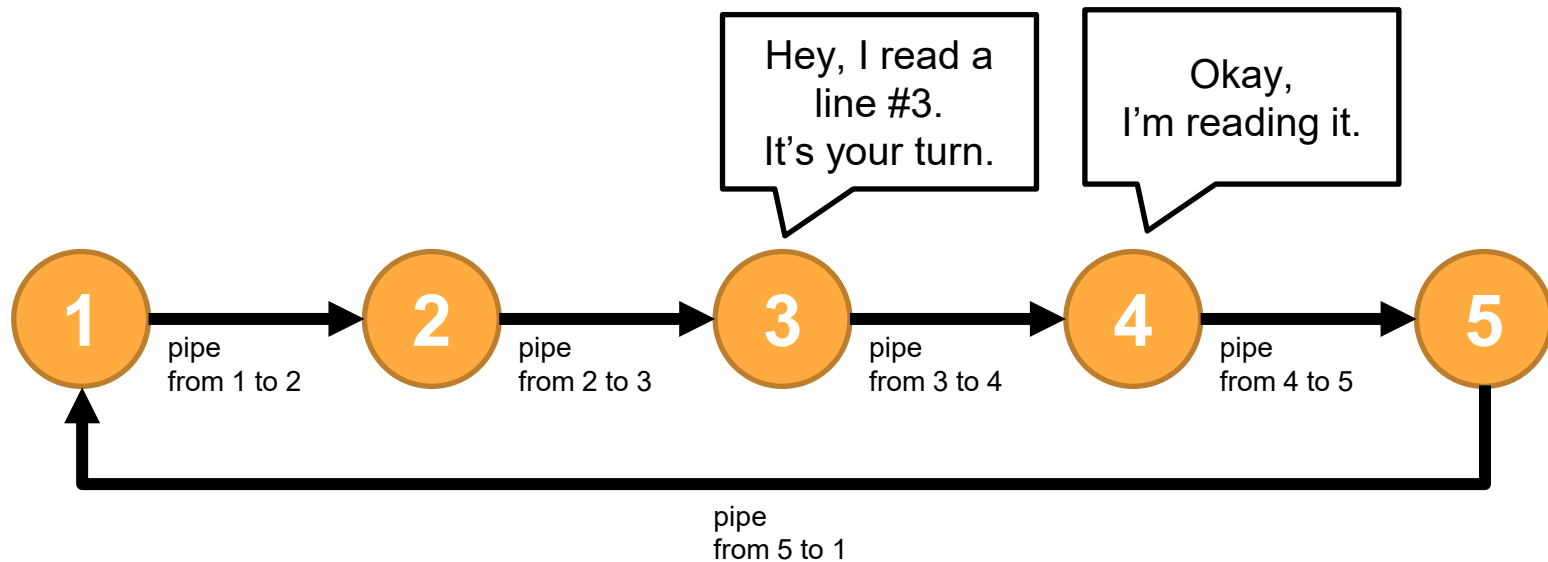
How can we let the next process read a line?

- PIPE!!



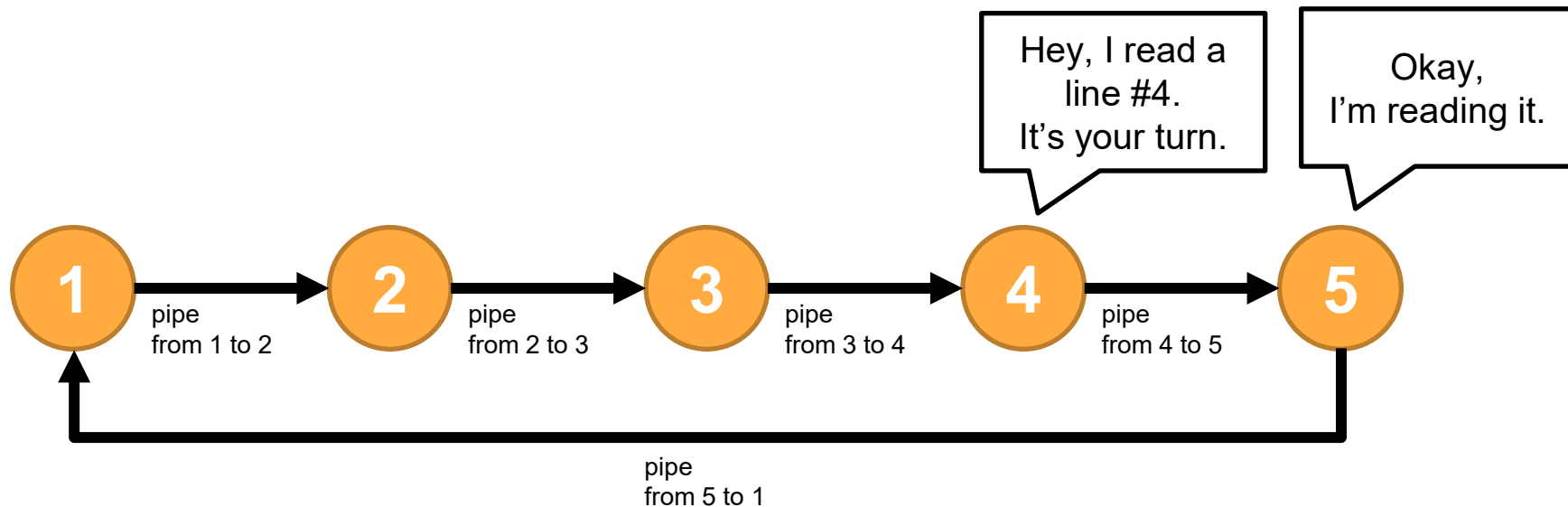
How can we let the next process read a line?

- PIPE!!



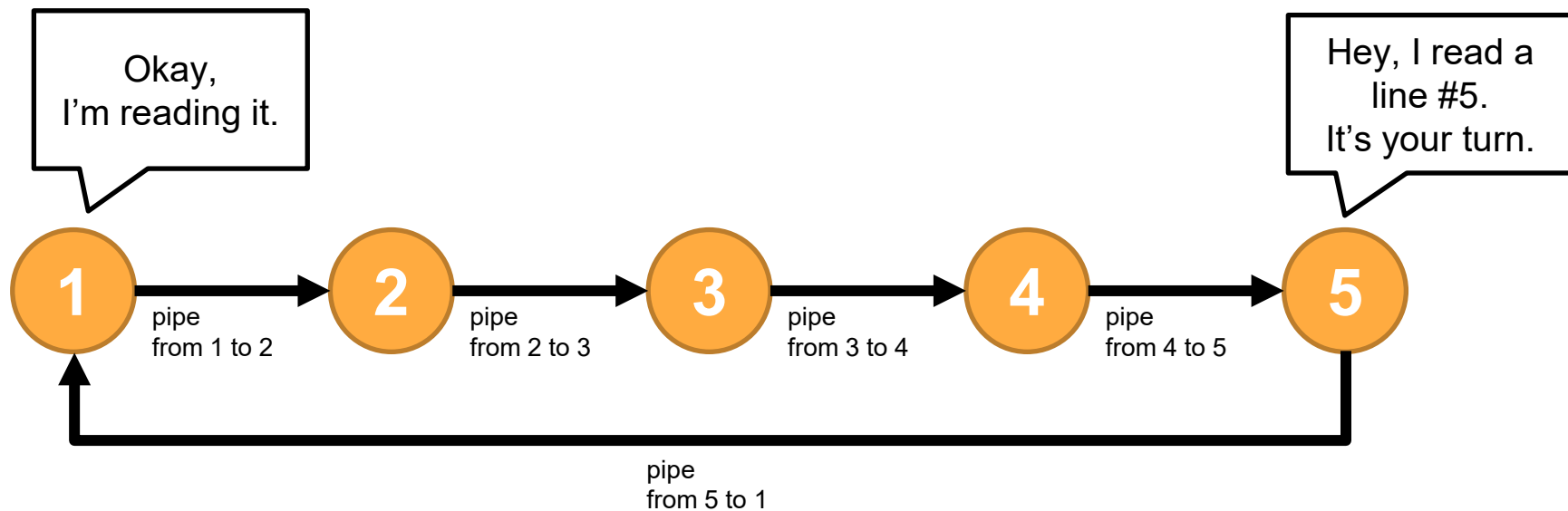
How can we let the next process read a line?

- PIPE!!



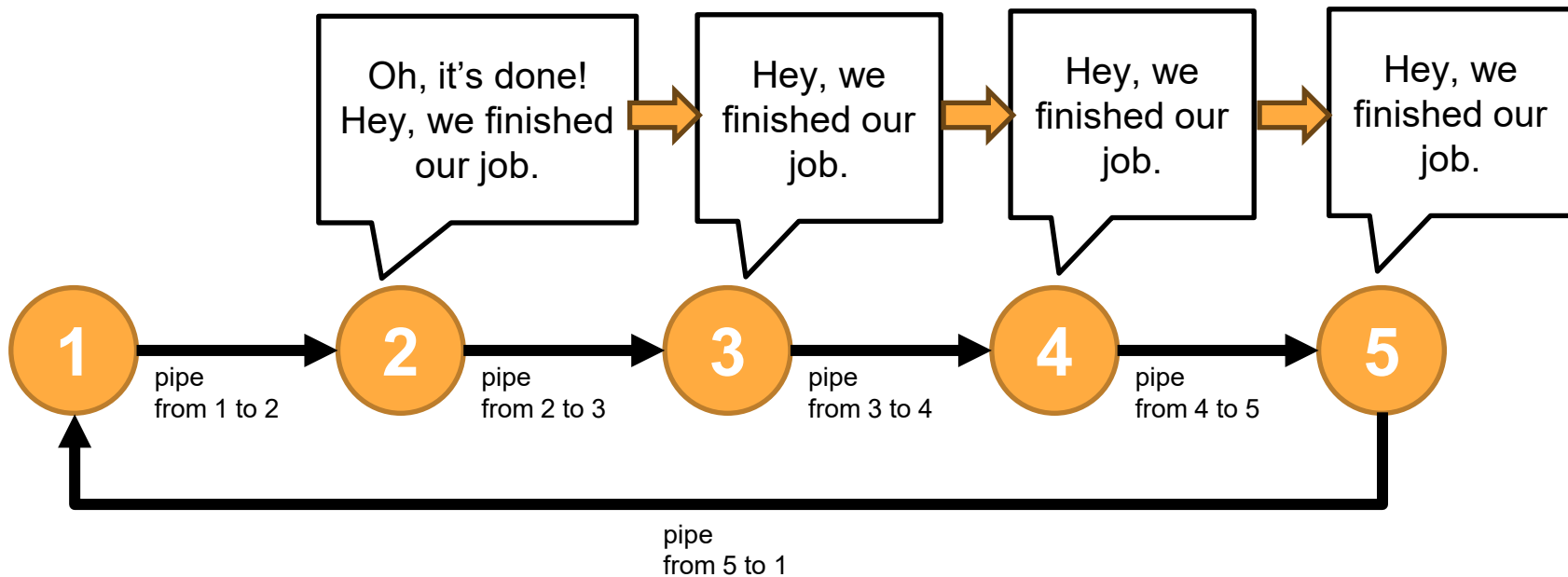
How can we let the next process read a line?

- PIPE!!



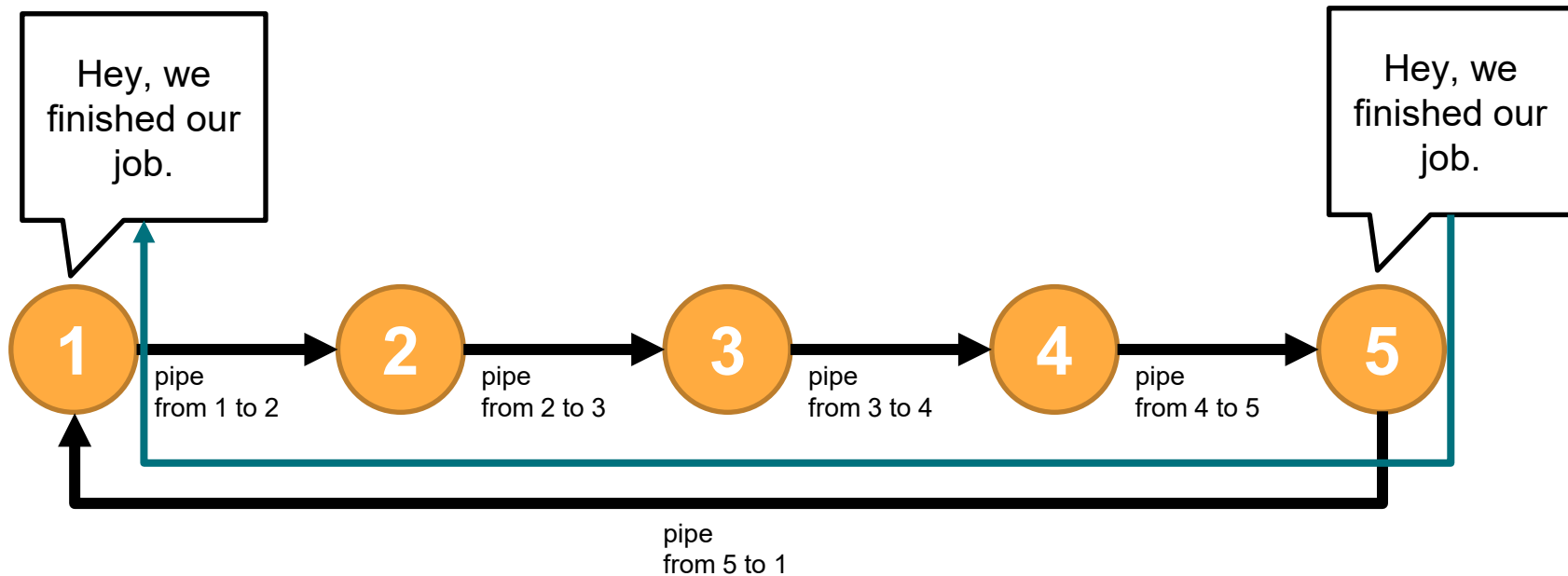
After reading ALL lines !!

- Exit processes from the last one.



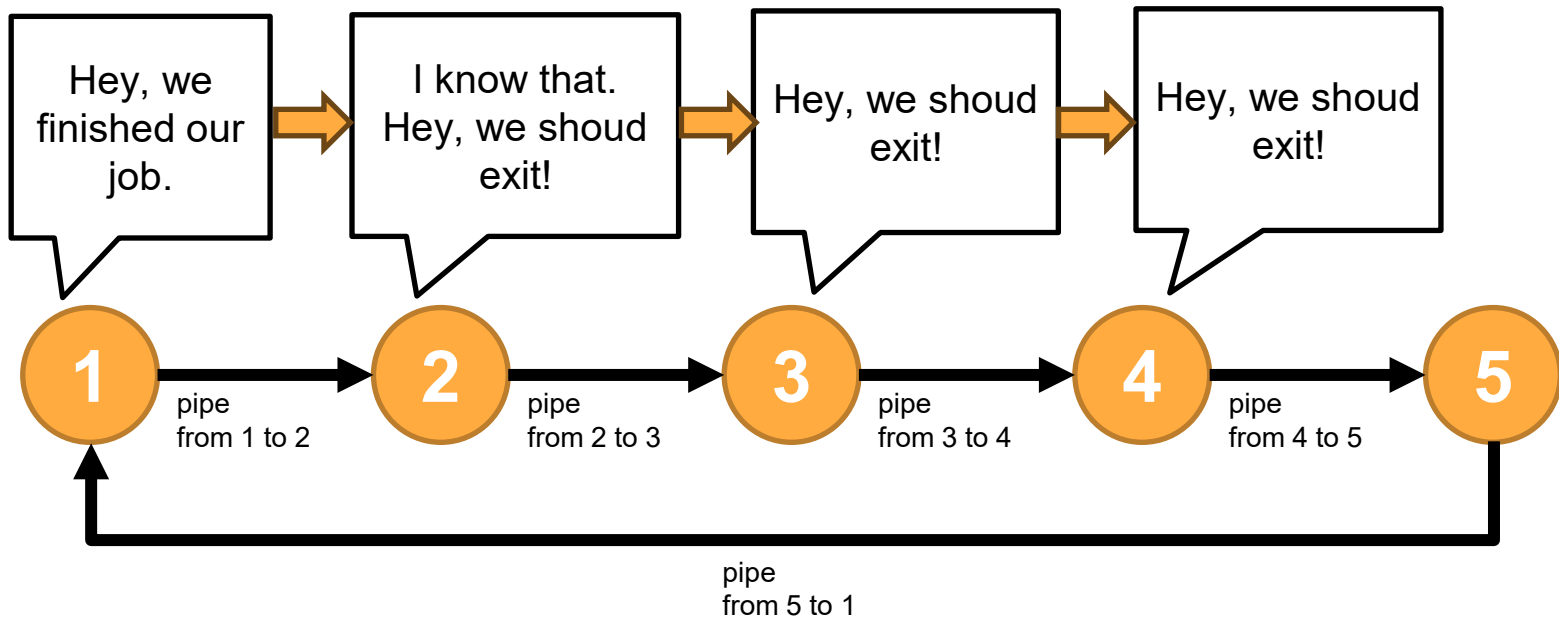
After reading ALL lines !!

- Exit processes from the last one



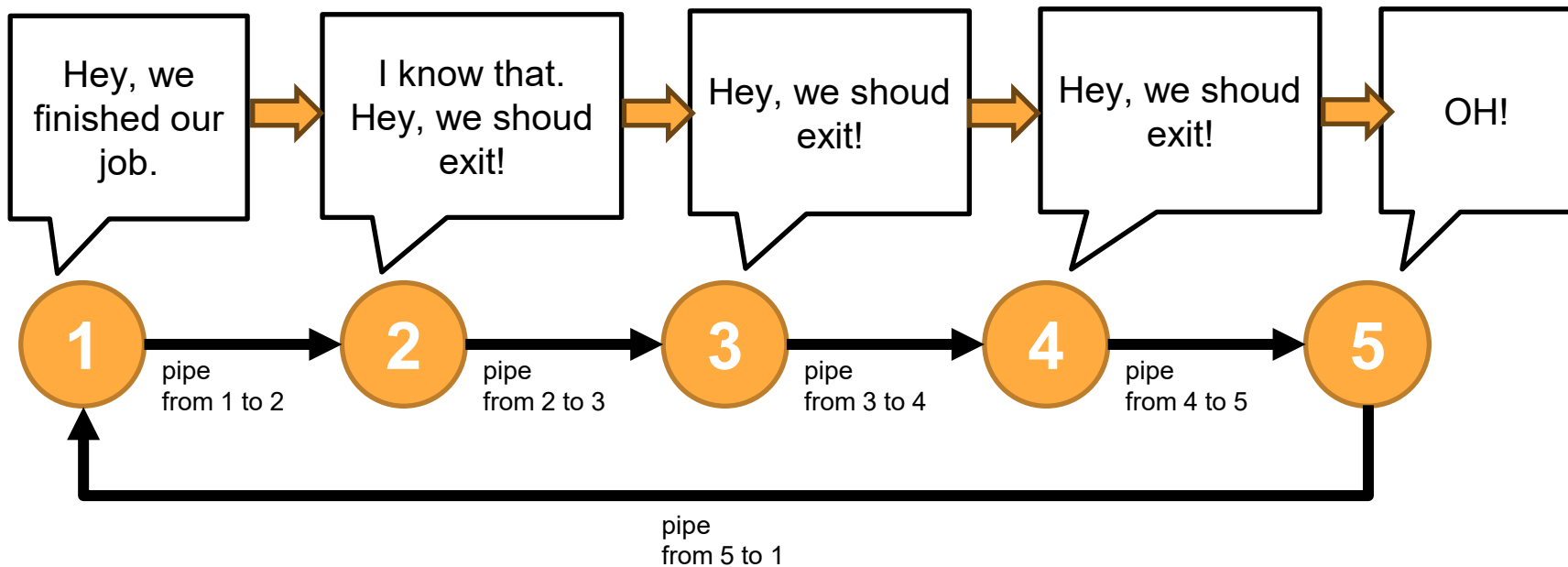
After reading ALL lines !!

- Exit processes from the last one.



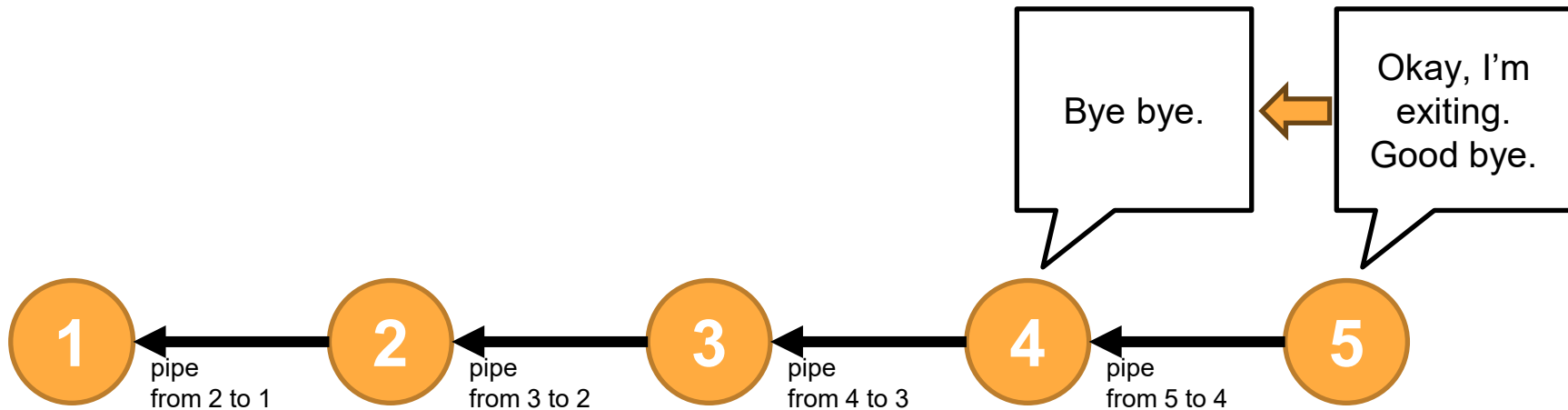
After reading ALL lines !!

- Exit processes from the last one.



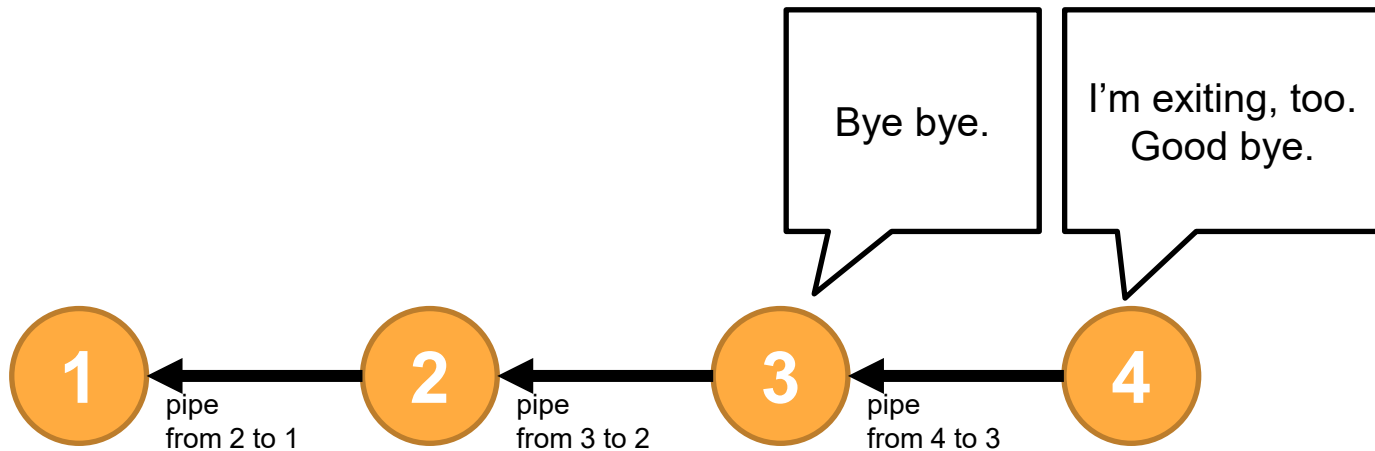
After reading ALL lines !!

- Exit processes from the last one.



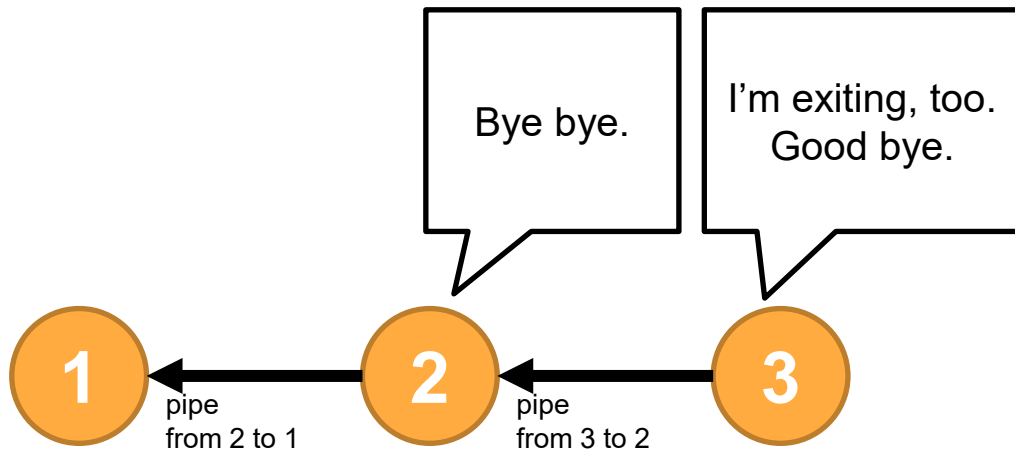
After reading ALL lines !!

- Exit processes from the last one.



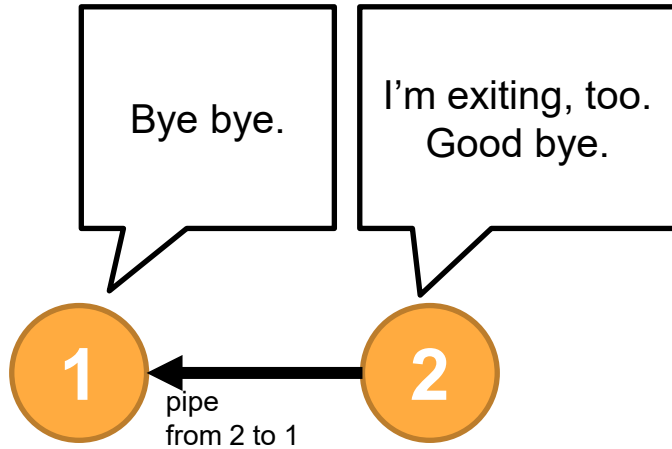
After reading ALL lines !!

- Exit processes from the last one.



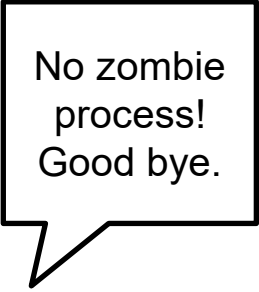
After reading ALL lines !!

- Exit processes from the last one.



After reading ALL lines !!

- Exit processes from the last one.



No zombie
process!
Good bye.



1



Why the parent process can't exit first?

Because a child process will be alone in the world.

If parent process exit first, child process becomes a zombie process.

Output format.

- Your program MUST print out each process's PID, #+line number and a line of text data
- After reading all lines, a process that read the last line of text data MUST print out its PID with a string "Read all data"
- Then, your program MUST exit from the last process, printing out PID with a string "I'm exiting..."

Example output of the program (1)

```
jsy01@s2lab02:~/Assignment3$ ./assignment3 input.txt 4
3078730 #1 One
3078731 #2 Two
3078732 #3 Three
3078733 #4 Four
3078730 #5 Five
3078731 #6 Six
3078732 #7 Seven
3078733 #8 Eight
3078730 #9 Nine
3078731 #10 Ten
3078731 Read all data
3078733 I'm exiting...
3078732 I'm exiting...
3078731 I'm exiting...
3078730 I'm exiting...
```

input.txt

```
1 One
2 Two
3 Three
4 Four
5 Five
6 Six
7 Seven
8 Eight
9 Nine
10 Ten
```

Example output of the program (2)

```
jsy01es2lab02:~/Assignment3$ ./assignment3 input.txt 16
3080130 #1 One
3080131 #2 Two
3080132 #3 Three
3080133 #4 Four
3080134 #5 Five
3080135 #6 Six
3080136 #7 Seven
3080137 #8 Eight
3080138 #9 Nine
3080139 #10 Ten
3080140 #11 Eleven
3080141 #12 Twelve
3080142 #13 Thirteen
3080143 #14 Fourteen
3080144 #15 Fifteen
3080145 #16 Sixteen
3080130 #17 Seventeen
3080131 #18 Eighteen
3080132 #19 Nineteen
3080133 #20 Twenty
3080133 Read all data
3080145 I'm exiting...
3080144 I'm exiting...
3080143 I'm exiting...
3080142 I'm exiting...
3080141 I'm exiting...
3080140 I'm exiting...
3080139 I'm exiting...
3080138 I'm exiting...
3080137 I'm exiting...
3080136 I'm exiting...
3080135 I'm exiting...
3080134 I'm exiting...
3080133 I'm exiting...
3080132 I'm exiting...
3080131 I'm exiting...
3080130 I'm exiting...
```

input.txt

```
1 One
2 Two
3 Three
4 Four
5 Five
6 Six
7 Seven
8 Eight
9 Nine
10 Ten
11 Eleven
12 Twelve
13 Thirteen
14 Fourteen
15 Fifteen
16 Sixteen
17 Seventeen
18 Eighteen
19 Nineteen
20 Twenty
```

Implementation Guideline

- (1) Create pipes to enable processes to communicate with each other by using `pipe()` system call.
- (2) Create a process based on the input arguments by using `fork()` system call.
- (3) Each process sequentially prints a line from text file with PID, line number. And pass the line number information to the next process through a pipe.
- (4) When all of lines in text file are printed out, print out “Read all data”. And pass the “Done” information to the next process.
- (5) All processes should exit with printing “I’m exiting...” from last one (when you exit a process, close all pipes that are opened).

TIPs

- You may use many system calls in this assignment including `fork()`, `pipe()`, `open()`, `read()`, `write()`, `close()`. We strongly recommend to familiar with these system calls before starting assignment.
- One of the challenging implementation is ensuring that the processes wait their turn. We recommend to use “blocking” characteristic. Please refer [pipe\(7\)](#).
- You can use other IPC, but we recommend to use pipe.
- This assignment is to implement a basic multi-process program, so don't worry about corner cases.

*pipe(7): <https://man7.org/linux/man-pages/man7/pipe.7.html>

Requirements

- Your program MUST use the number of processes received as an argument.
- PID of parent process and PID of child process can be discontinuous.
- Your program MUST not create Zombie processes.
- In our input file, there exists newline('\n') at the end of file.
*'\$' means newline('\n')

```
jsy01@s2lab02:~/Assignment3$ cat -A input.txt
One$
Two$
Three$
Four$
Five$
Six$
Seven$
Eight$
Nine$
Ten$
jsy01@s2lab02:~/Assignment3$
```

Submission

- You should submit your code with a code description that explains your code (i.e., comments in the file). In the description, your code must be well commented to explain your algorithm. Make your code .zip file with "StudentID_YourName.zip" and submit your .zip file on blackboard.

ex)

20205147_HongjunYang.zip

- assignment3.c // assignment3 code
- assignment3.h // assignment3 header file
- Makefile // Makefile
- report.pdf // assignment3 code description

For the report, please write it briefly.

More specifically, It is enough to copy and paste the important part of your source code and briefly describe it.

If there are some websites or codes that you refer to, it must be included in your report to avoid code plagiarism.