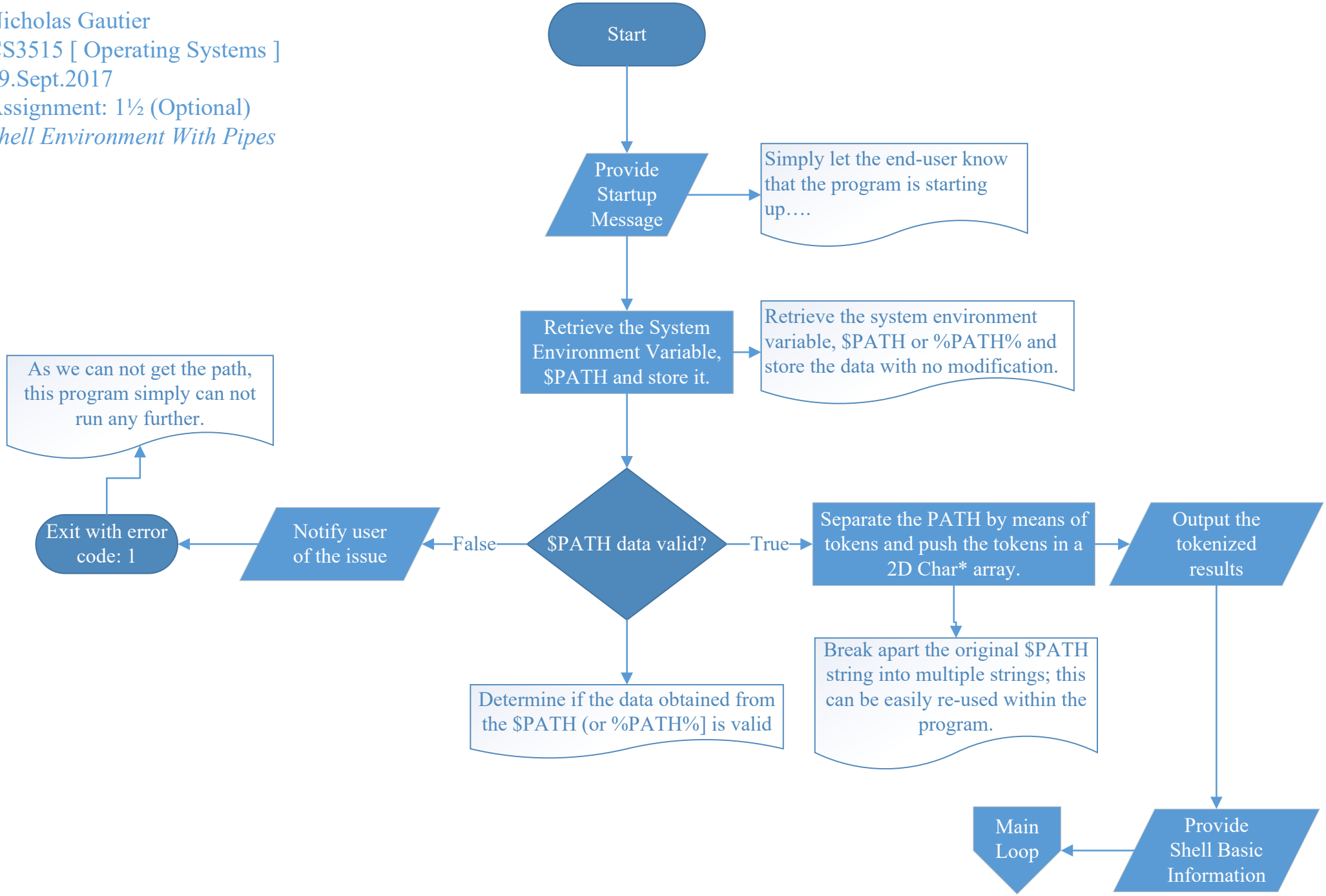
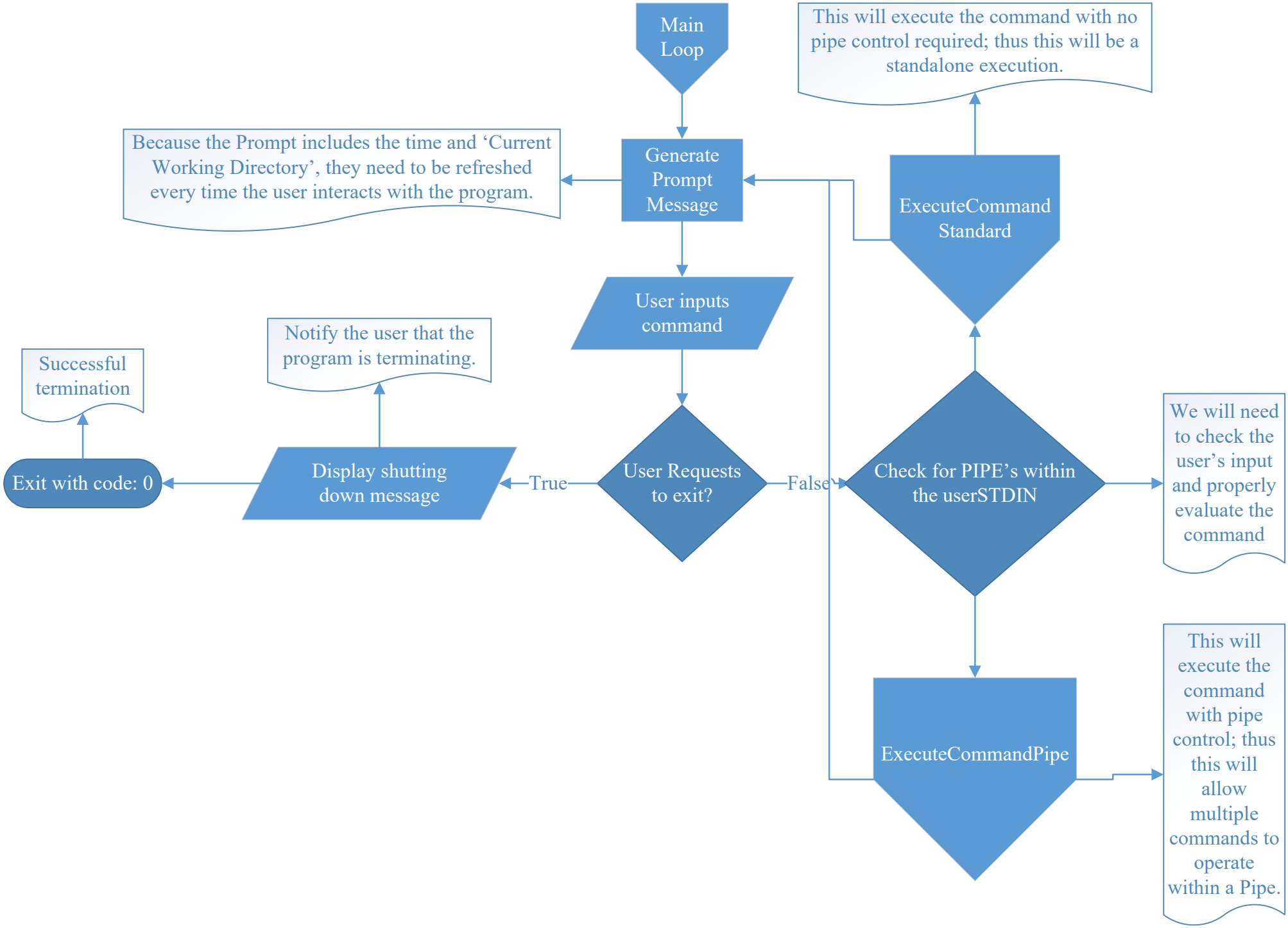
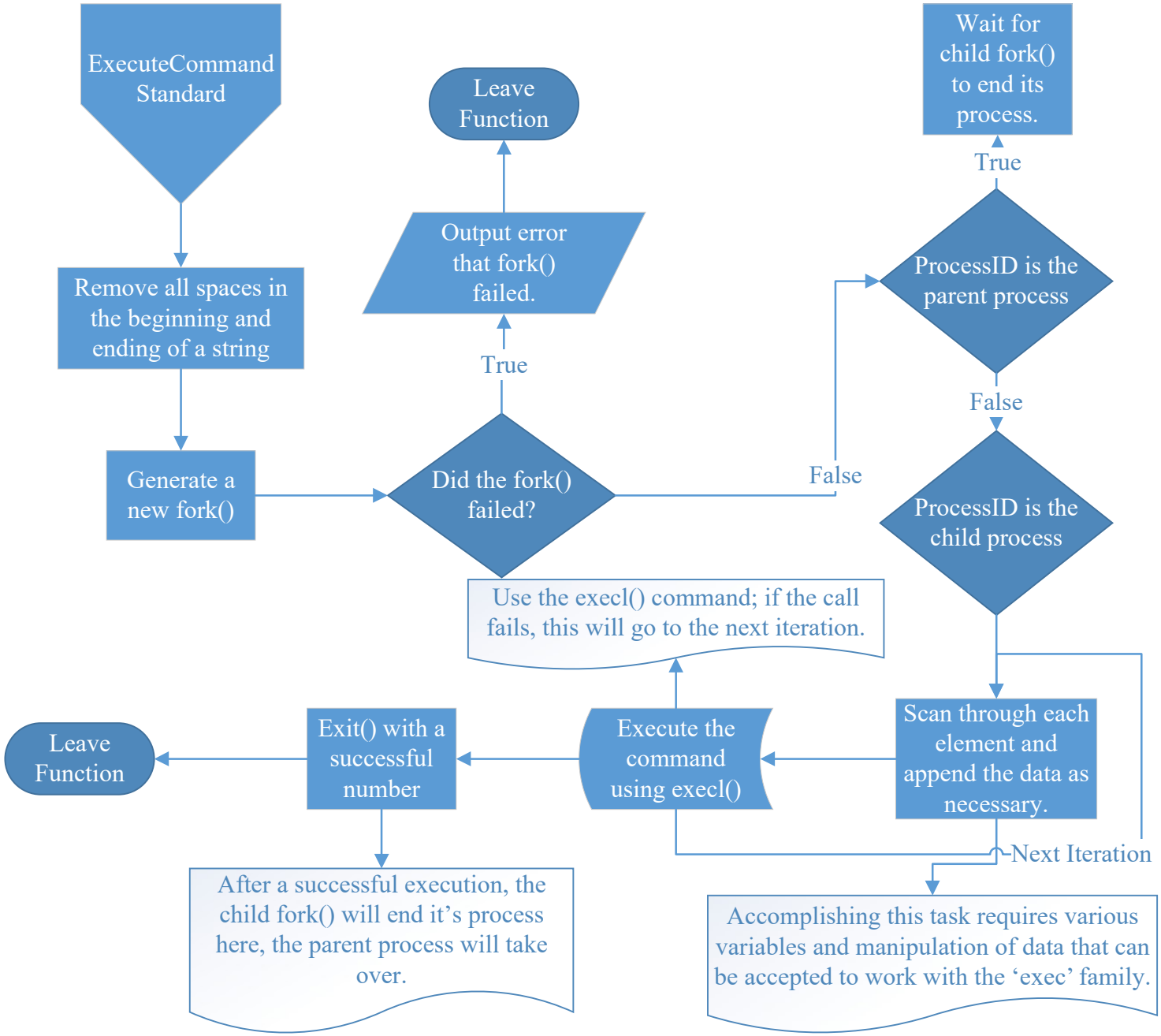
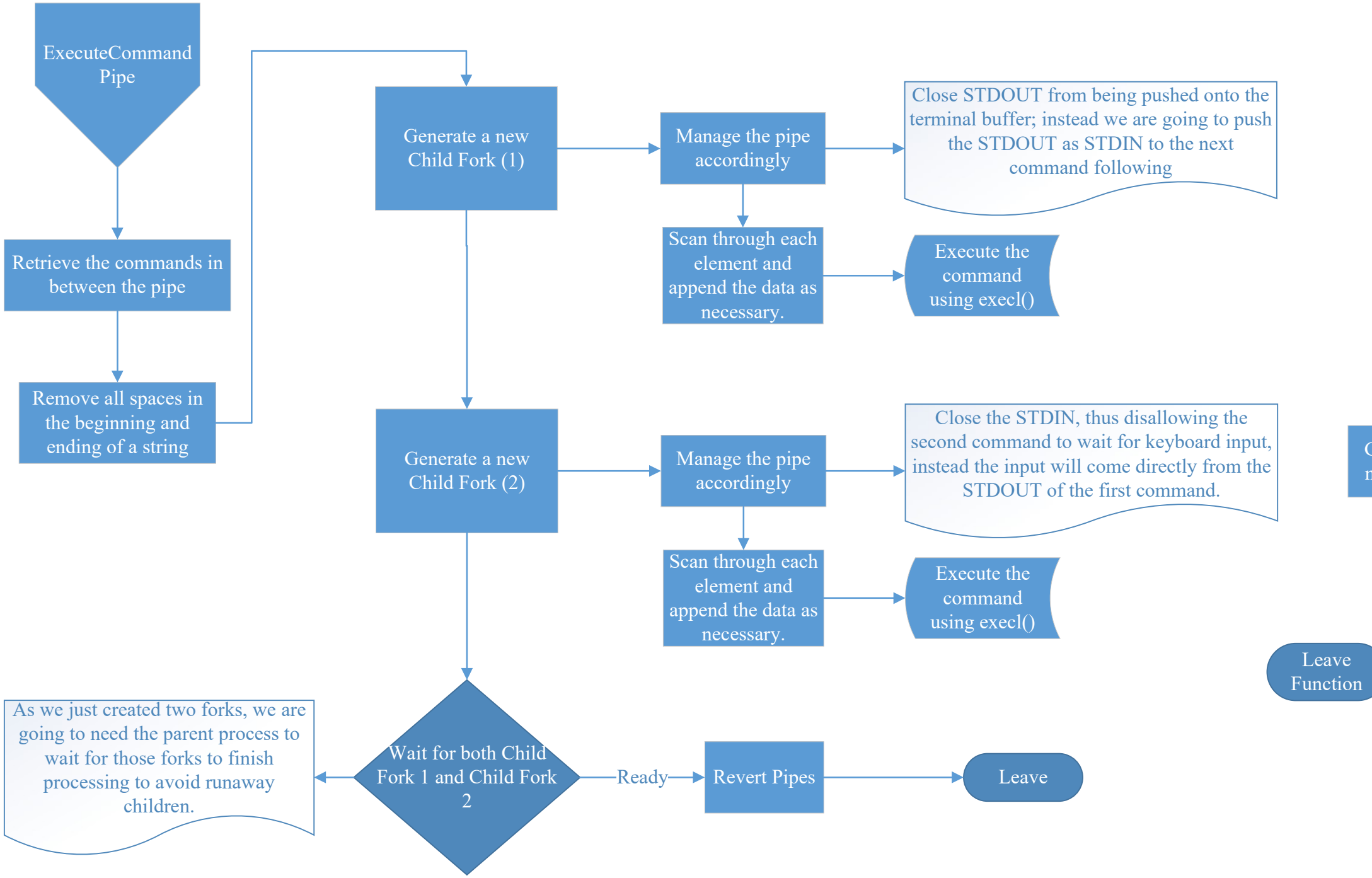
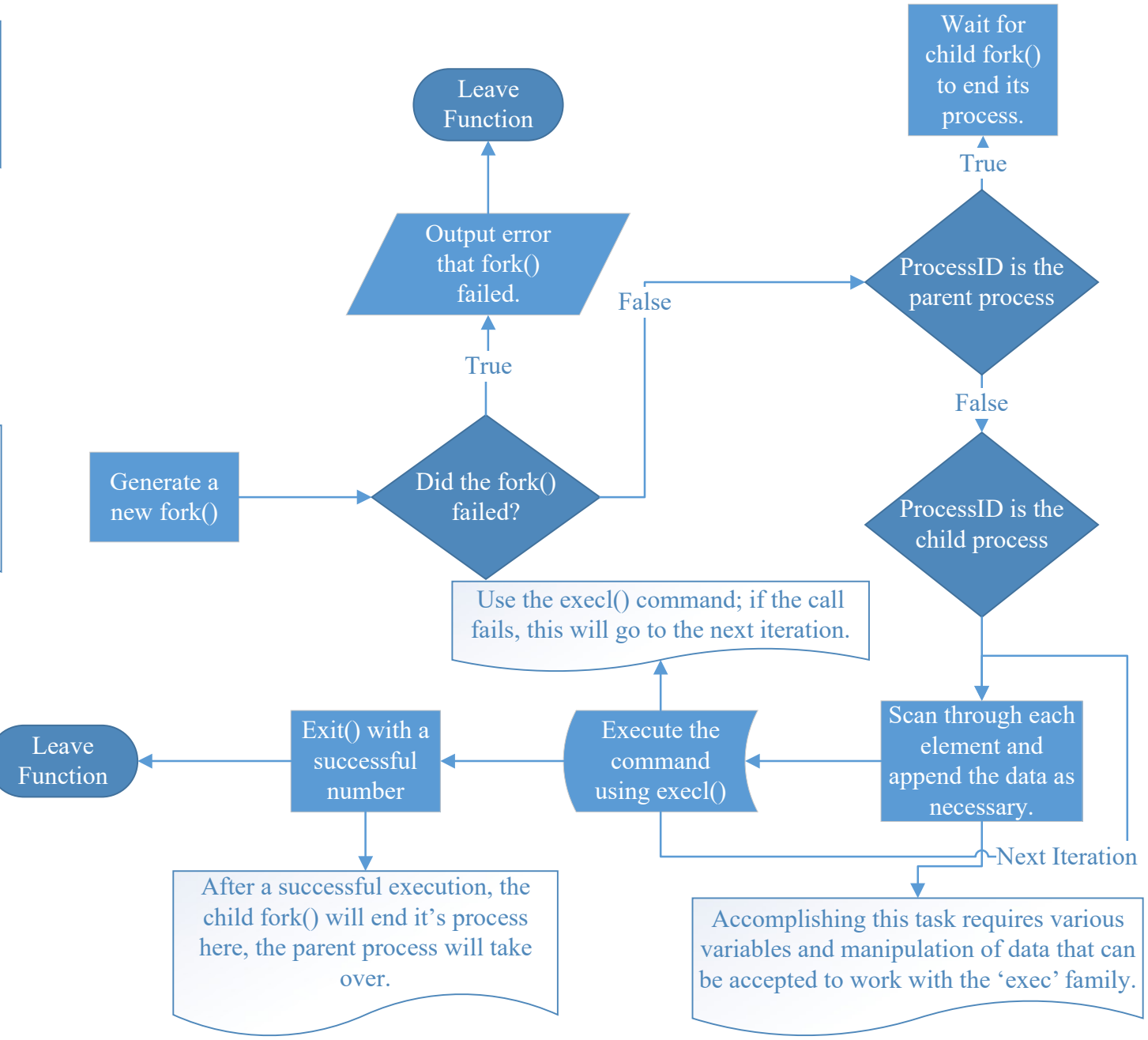
Nicholas Gautier
CS3515 [Operating Systems]
19.Sept.2017
Assignment: 1½ (Optional)
Shell Environment With Pipes




Main Loop
Generate Prompt Message
Because the Prompt includes the time and 'Current Working Directory', they need to be refreshed every time the user interacts with the program.
User inputs command
User Requests to exit?
True
Display shutting down message
Notify the user that the program is terminating.
Exit with code: 0
Successful termination
False
Check for PIPE's within the userSTDIN
We will need to check the user's input and properly evaluate the command
ExecuteCommand Standard
This will execute the command with no pipe control required; thus this will be a standalone execution.
ExecuteCommandPipe
This will execute the command with pipe control; thus this will allow multiple commands to operate within a Pipe.


ExecuteCommand Standard
Remove all spaces in the beginning and ending of a string
Generate a new fork()
Did the fork() failed?
True
Output error that fork() failed.
Leave Function
False
ProcessID is the parent process
True
Wait for child fork() to end its process.
False
ProcessID is the child process
Scan through each element and append the data as necessary.
Execute the command using execl()
Use the execl() command; if the call fails, this will go to the next iteration.
Exit() with a successful number
Leave Function
After a successful execution, the child fork() will end it's process here, the parent process will take over.
Next Iteration
Accomplishing this task requires various variables and manipulation of data that can be accepted to work with the 'exec' family.


ExecuteCommand Pipe
Retrieve the commands in between the pipe
Remove all spaces in the beginning and ending of a string
Generate a new Child Fork (1)
Manage the pipe accordingly
Close STDOUT from being pushed onto the terminal buffer; instead we are going to push the STDOUT as STDIN to the next command following
Scan through each element and append the data as necessary.
Execute the command using execl()
Generate a new Child Fork (2)
Manage the pipe accordingly
Close the STDIN, thus disallowing the second command to wait for keyboard input, instead the input will come directly from the STDOUT of the first command.
Scan through each element and append the data as necessary.
Execute the command using execl()
Leave Function
As we just created two forks, we are going to need the parent process to wait for those forks to finish processing to avoid runaway children.
Wait for both Child Fork 1 and Child Fork 2
Ready
Revert Pipes
Leave


Leave Function
Wait for child fork() to end its process.
True
Output error that fork() failed.
False
ProcessID is the parent process
True
False
Generate a new fork()
Did the fork() failed?
ProcessID is the child process
Use the execl() command; if the call fails, this will go to the next iteration.
Leave Function
Exit() with a successful number
Execute the command using execl()
Scan through each element and append the data as necessary.
Next Iteration
After a successful execution, the child fork() will end it's process here, the parent process will take over.
Accomplishing this task requires various variables and manipulation of data that can be accepted to work with the 'exec' family.