Test Data - nonohup, whatIf

How the test data was partitioned

- nonohup and whatIf are given individual test cases to fit three of scenarios according to equivalence partitioning.
  - Edge cases / wrong cases: No arguments or too many arguments are passed to the syscall usually invoking an incorrect behavior.
  - Regular cases: With all different arguments
  - Combination cases: The behavior of the system call being tested was combined with ;,&,| to see their behavior. nonohup and whatIf are also called through each other to make sure they work with each other

These partitionings make sure that all the cases are being considered for each of the system calls and ensures complete test coverage.

TEST CASES

nonohup

Test cases - Outputs - Description

Test Cases	Output	Description
$ nonohup	$ exec: fail zombie!	This is the base case where we are passing nonohup without any command or arguments along with it. When that is executed, the prompt enters a new line and then the failure message is printed.
$ nonohup nonohup echo a	exec: fail $S20 zombie!	runcmd EXEC searches for "nonohup" in the the argv and then executes the rest of the command. Since 2nd nonohup is not checked for, it fails
$ nonohup echo a	$S20 a zombie!	Second prompt is displayed, then the command and args after nonohup are executed.
$ nonohup echo a &	zombie! $S20 a zombie!	& and nonohup serve the same function so the zombie is called twice since there are 2 forks() that are not being reaped.
$ echo a; nonohup echo b; echo c	a b zombie! c	echo a is executed, nonohup echo b is executed in the background but not reaped (hence the zombie), echo c is executed
$ echo a &;nonohup echo b &	$S20 a zombie! zombie! b zombie!	& invokes a zombie! due to unreaped child process. nonohup also invokes a zombie due to unreaped child
$ echo a | nonohup echo b	exec: fail exec $b zombie! failed	echo a is executed and then a zombie is called because a child process isn't reaped.

whatIf

Test cases - Outputs - Description

Test Cases	Output	Description
$ whatIf	Enter your content: hi my name is ritwik	Prompt enter your content shows up to help the user to type in something until it encounters `control + d`. Now running cat parent.txt and child.txt, you should see the same file. wc should give the accurate word count of the file.

Test Cases	Output	Description
$ whatIf sample.txt	$S20	The file sample.txt is copied line by line to parent.txt and child.txt. cat into parent.txt and child.txt should show you the content and give you accurate word length
$ whatIf wrongFile.txt	Failed to find the file wrongFile.txt	When a random file is passed, it returns the expected behavior of letting the user know that the file is invalid.
$ whatIf parent.txt / whatIf child.txt	$S20	The contents of the files don't change since child.txt and parent.txt have the same content.
$ whatIf parent.txt & whatIf child.txt	$S20	Performs like expected where the parent.txt and child.txt are written to each other.
$ whatIf parent.txt ; whatIf child.txt	$S20	Performs as predicted where both the commands run properly and write information onto themselves.
$ nonohup whatIf	$ Enter Content:	This starts execution in the background but doesn't take any input from the user because the child process has been killed.
$ nonohup whatIf sample.txt	$S20 zombie!	Zombie is displayed because the child process hasn't been reaped but whatIf is successfully executed in the background! cat parent.txt and cat child.txt to see the effects.
$ whatIf & whatIf sample.txt	$S20 zombie! Enter content:	whatIf sample.txt has been executed but whatIf in the background doesn't take any input