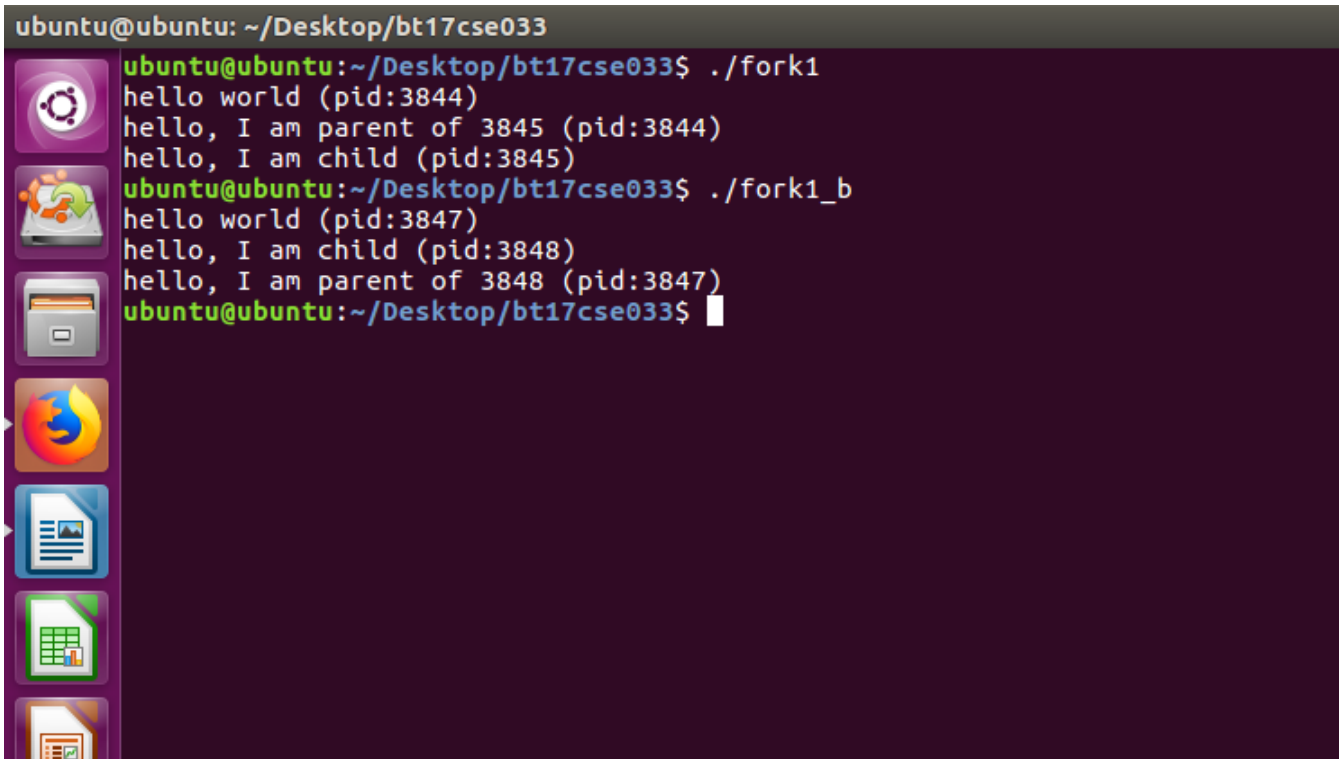
OS Lab_3

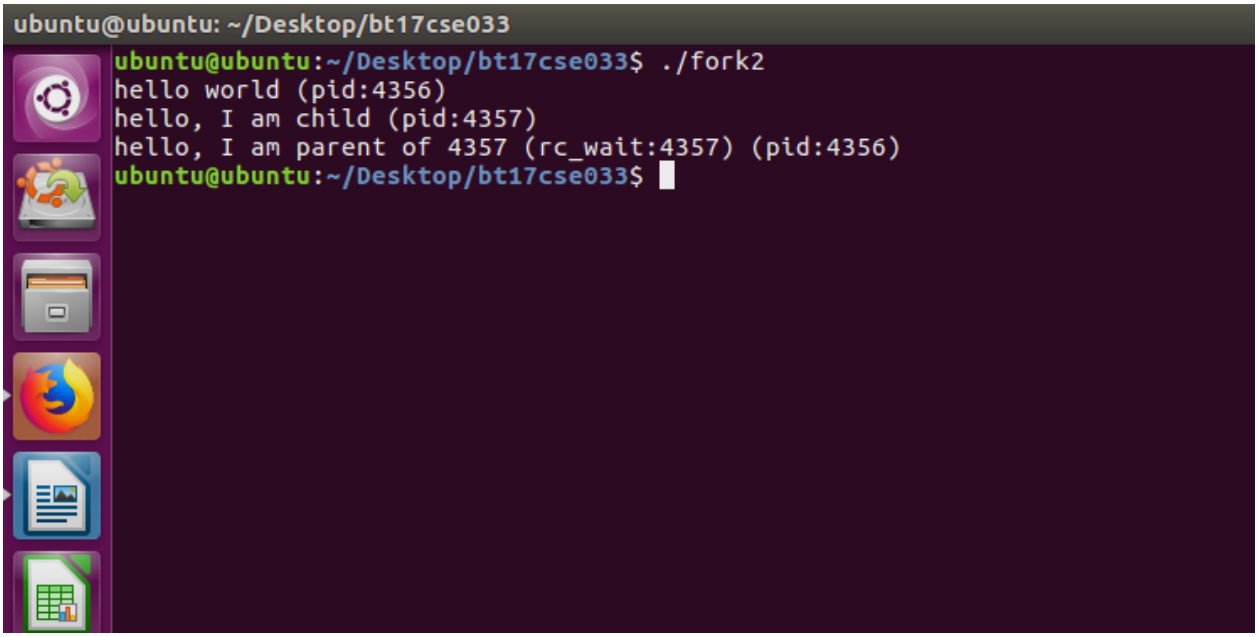
Question 1: FORK1
Answer:
Here, after fork() system call, printing statement of parent is executed first, and then of child. But it's not necessary, as both the processes are running concurrently (Absence of wait() system call), upon different executions, order of printing statements (the order of execution of child and parent process) might differ form this, and either can be executed first.

But however, if parent process is taking a lot of time(can be checked by running a big loop inside a parent process), output of child process will come first, as cpu scheduling will be done. While child process might finish it's execution as soon as cpu is assigned to it, parent process can still be running as cpu is not given to only one process, for as long as it demands, in case of preemptive scheduling.



Here, fork1 gives output of FORK1 code provided, while fork1_b gives output of modified code with a big loop inside parent process.

Question 2: FORK2
Answer:
Here, after a fork() system call, child process is executed first.
A wait() system call has been used inside the code.
In case the process is parent process, wait() system call waits for the child process to execute, then only parent process is executed. So child's print statement is printed first and then parent's print statement.



In this case, using a big loop inside child process is also not going to alter the order as wait() system call will anyhow wait for child to get executed.

Question 3: FORK3

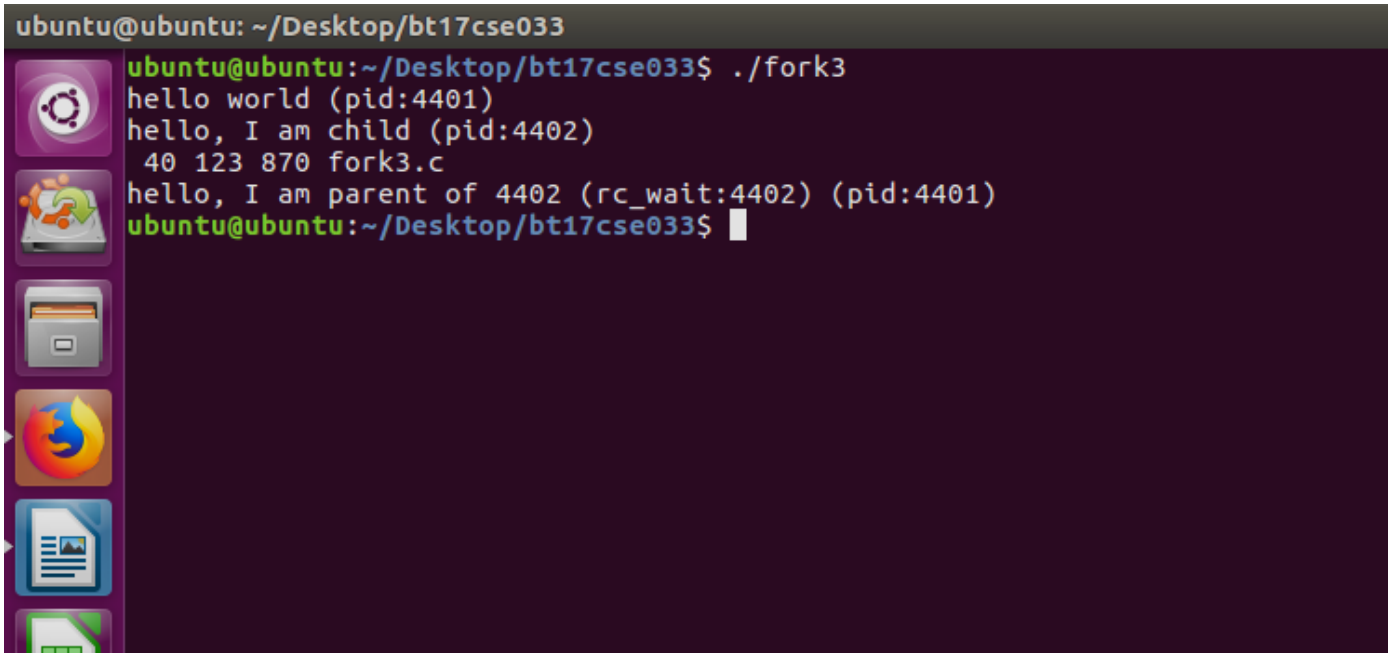
Answer:

In this code, a fork() statement is used after a printing operation, so first line of output is result of first printf statement of parent process.

After this, there is a wait() system call used in the parent process, so parent process is executed only after child is executed completely.

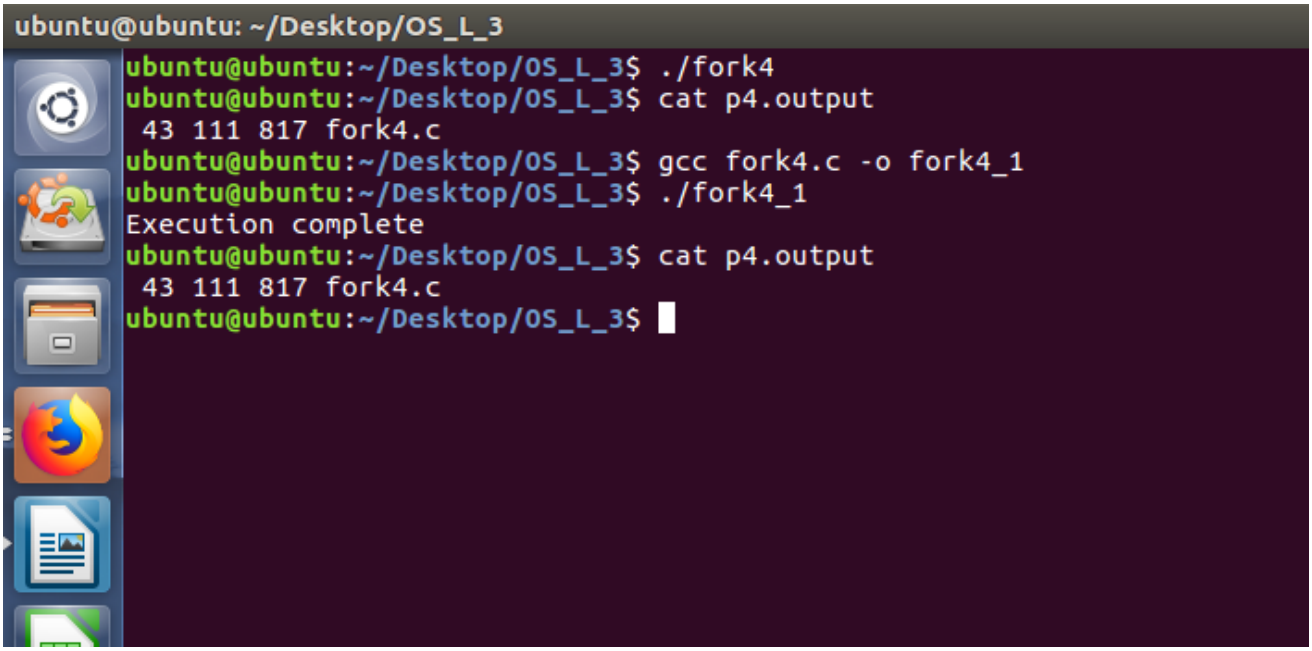
Inside child process, first an informatory printing statement is executed(Output line 2), and then current child process is replaced with a new process which performs word count (‘wc’ command) operation on file ‘fork3.c’ using *execvp (Replaces current process with a new process)* command, and output of this file is printed on terminal (Output line 3).

Now that the child process has ended, now a informatory statement from the parent process is printed (Output line 4).



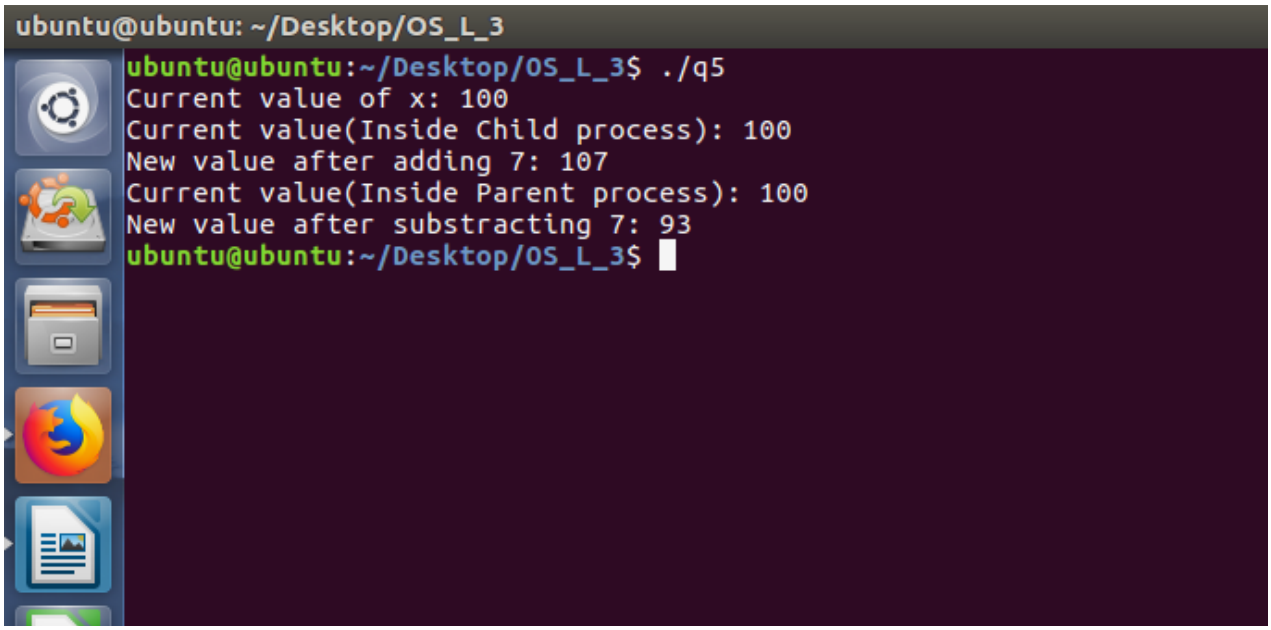
Question 4: FORK4
Answer:
No print statements are used in this program.
A fork() system call is used which creates a new child process.
Inside parent process, wait() system call is used, so it waits for child process to get executed first.
Inside child process, first default file descriptor for output, STDOUT_FILENO, is closed, so that no output is printed on terminal, instead output of the code is directed to file ‘p4.output’ using open() system call.
This child process is replaced by a new process that does word count operation (‘wc’) on file ‘fork4.c’ using execvp() system call, and its output is directed to the file ‘p4.output’.
After child’s process execution, parent completes its execution.

Output can be seen by reading ‘p4.output’ file.



Only output of child process was directed to file, this can be checked by putting a printf statement in parent process, which will print output on terminal.

Question 5: Program
Output:



Initial value of x in both child and parent process remains same.
However changing value of x in one doesn't changes value of other x.

Question 6: Program
Output:

Initial value of x in both child and parent process remains same.
However changing value of x in one doesn't changes value of other x.

Waitpid() is used generally to wait until a process finishes, based on its process ID (pid). It can be used to wait for any of a group of child processes, either one from a specific process group or any child of current process.

In this code, waitpid is used to stop execution of parent process until the child process of pid returned to parent process during fork() process is finished.