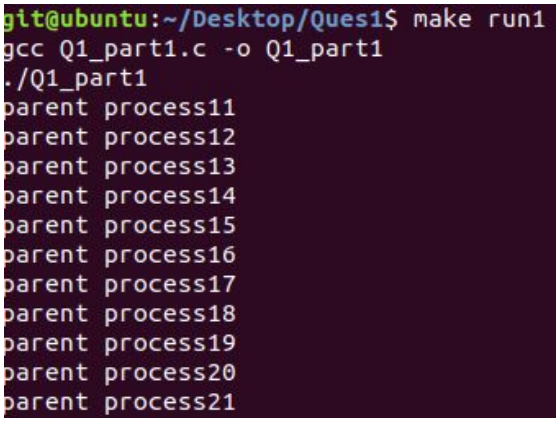
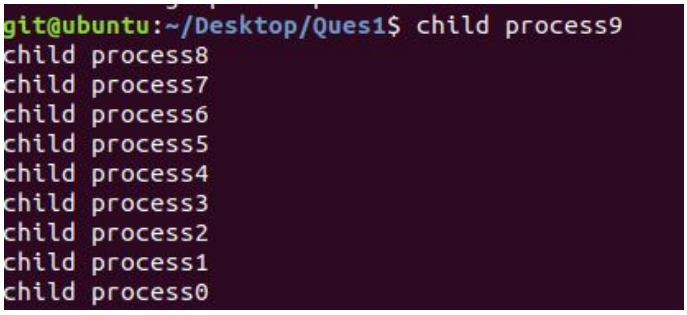
Gitansh Raj Satija OS Assignment2 Ques1

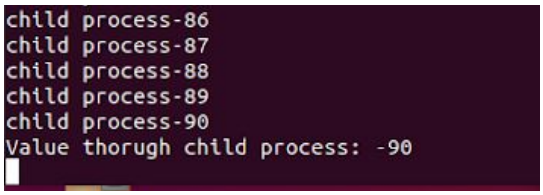
Fork:

The fork system call creates a new process (complete copy/clone of all the states). The new process has different data/state. Hence, the changes in the global variable of child or the parent process would not affect the state of the other variable in the other process and so the change would not be seen in the other.

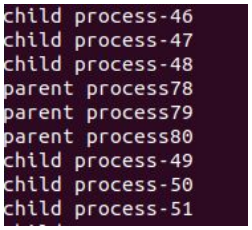
As you can see in the following 3 snippets, in this case the parent process runs first (as preferred by the system os) and starts incrementing the value of the global variable. When it reaches 100 the parent process modification to the global variable is over. As you can see the child process does not share the value of the global variable with the parent process, and takes the original value i.e. 10 and starts decrementing it till -90.





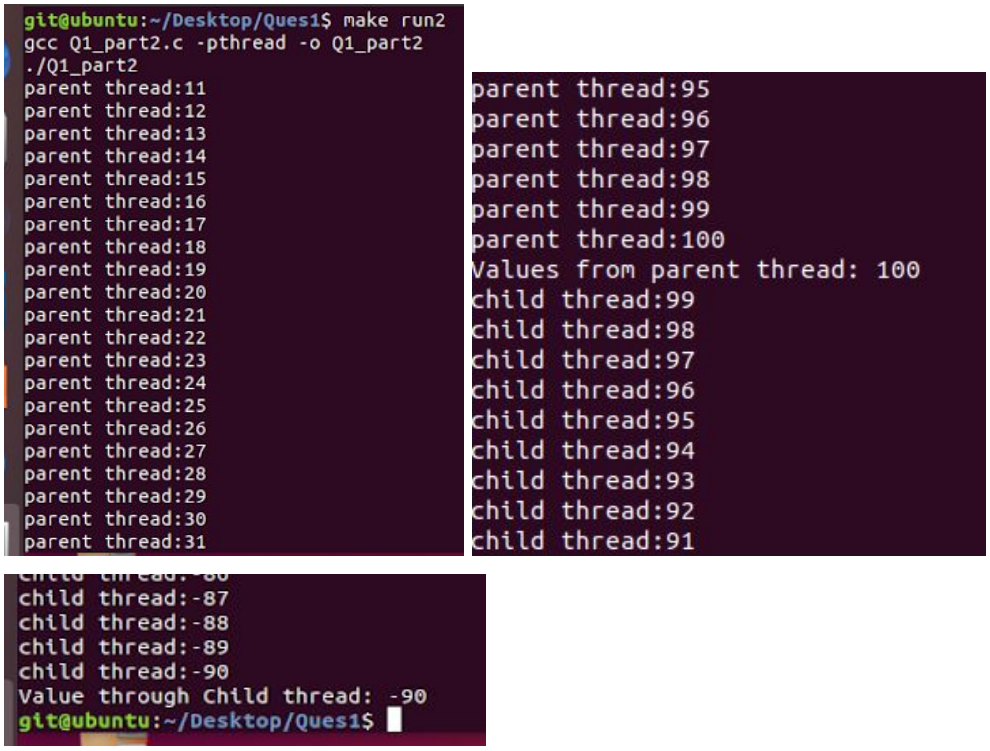


However, it is important to note that parent and child processes run in parallel by context switching which can be displayed in another output using the same code. As you can see, there is context switching between the child and parent process, both of which have their own value of the global variable x. (while for child process x is -48 , for parent process it is 78 hence showing that x is different for both.)



Thread:

Thread is a segment of a process and does not create a whole new process. Threads share the same memory and open files. Within the shared memory, each thread gets its own stack, instruction pointer and registers. As said, since memory is shared, there is no memory protection among the threads in the process. Therefore, the value of the global variable is shared between both the process, therefore any change in the value of the global variable in one thread affects in the other thread. Here in the following 3 snippets you can see that the parent thread is executed first and then the child thread. But the difference this time is that the value of x is the same for both the threads as they share this memory (of global variables). Hence when child thread executes in this example it starts decrementing after 100 and not the initial value of x that was 10.



However in the output of the same code on the left you can see context switching between the parent thread and the child thread by the os and as you can see child thread takes the value of x which is just after its last increment from the parent thread indicating shared memory.

In both the cases (fork and thread) one reason why context switching occurs is that during incrementing we are printing the value at each stage, printf being a write system call, it goes to the scheduler which either executes it or can block it and switch context to the other process/thread. Note: Error handling using perror has been done for both the c programs for system calls like fork, pthread_create, pthread_join, scanf,printf. Appropriate error messages are returned in case any error occurs.