Operating Systems Assignment 2 Question 1
Yatharth Taneja | 2019346

How to run the code.
Use command 'make run1' to run part 1 i.e the fork()
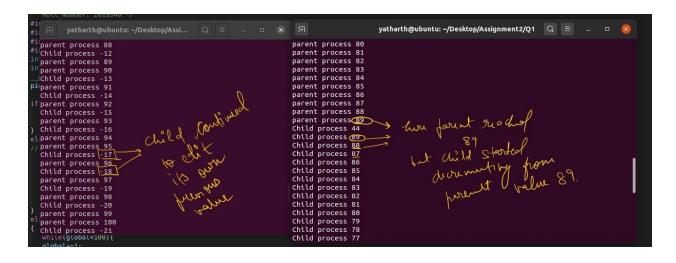And 'make run2' to run part 2 i.e the pthread_create()

fork()
- It makes duplicates of all processes memories.
- Since both, the process will have identical but separate address spaces variables announced before the fork() will have the same initial values but if the parent changes the value of its variable, the modification will only affect the variable in the parent process's address space. Other address spaces created by fork() calls will not be affected even though they have identical variable names.

Note: We can use waitpid() to wait for the child to complete before returning 0 from main (parent process) but sir asked us not to use it. Hence there might be a case in which the current process is terminated as soon as the parent reaches 100. The child process becomes a zombie process and is allocated a parent internally and it is completed in the next cycle.

pthread_create()
- It makes a partial copy of the processes memories
- Threads are not independent of one other like processes as a result threads share with other threads their code section, data section, and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.
- Unlike the fork() process, the variables shared between threads are not independent to each other and hence any changes in any thread will result in changes in all necessary threads too.

```
parent process 88
Child process -12
parent process 89
parent process 90
Child process -13
parent process 91
Child process -14
parent process 92
Child process -15
parent process 93
Child process -16
parent process 94
parent process 95
Child process -17
parent process 96
Child process -18
parent process 97
Child process -19
parent process 98
Child process -20
parent process 99
parent process 100
Child process -21
while(global<100){
global+=1;
```

```
parent process 80
parent process 81
parent process 82
parent process 83
parent process 84
parent process 85
parent process 86
parent process 87
parent process 88
parent process 89
Child process 44
Child process 89
Child process 88
Child process 87
Child process 86
Child process 85
Child process 84
Child process 83
Child process 82
Child process 81
Child process 80
Child process 79
Child process 78
Child process 77
```

(handwritten) child continued to edit its own previous value

(handwritten) here parent reached 89 but child started decrementing from parent value 89.

fork()   **pthread_create()**

From the above screenshot we can see that in process fork() the child is decreasing the value of global that it copied hence the highlighted -17 is decremented to -18 simultaneously parent is incrementing the value of global variable it copied to it's address process hence both the process change the variable that is in their own address space

While on the thread we can see that the parent process had the value of global variable as 89 and the child process was writing a value of 44(since both process are reading and writing at the same address space therefore there is data race.) to the terminal but since the parent changed the value of the global variable to 89 in the next print child process starts decrementing the new value 89 to 88 , 87 …

In the fork process child value will vary from (-90 to 10) and parent (10 to 100)
While in pthread the values of parent and child can range from (-90 to 100)