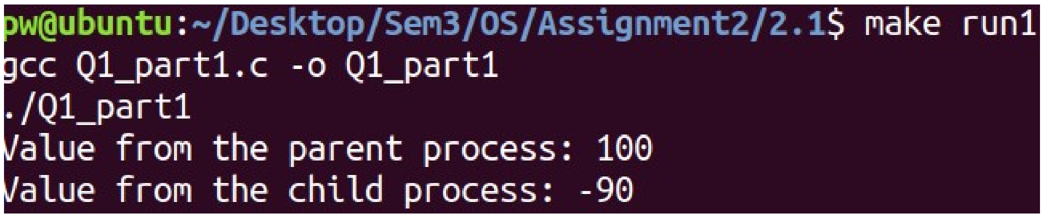
CSE 231: OS Assignment 2

Differences Between Processes and Threads

1 Fork

- It creates a whole new process.
- The new process has its own memory and variables.
- Change in variables of any of the parent or child variables do not affect the variables of the other.
- Since both the processes have their own separate variables and memory spaces, if any one of these changes the global variable, the change would not get reflected in the other.

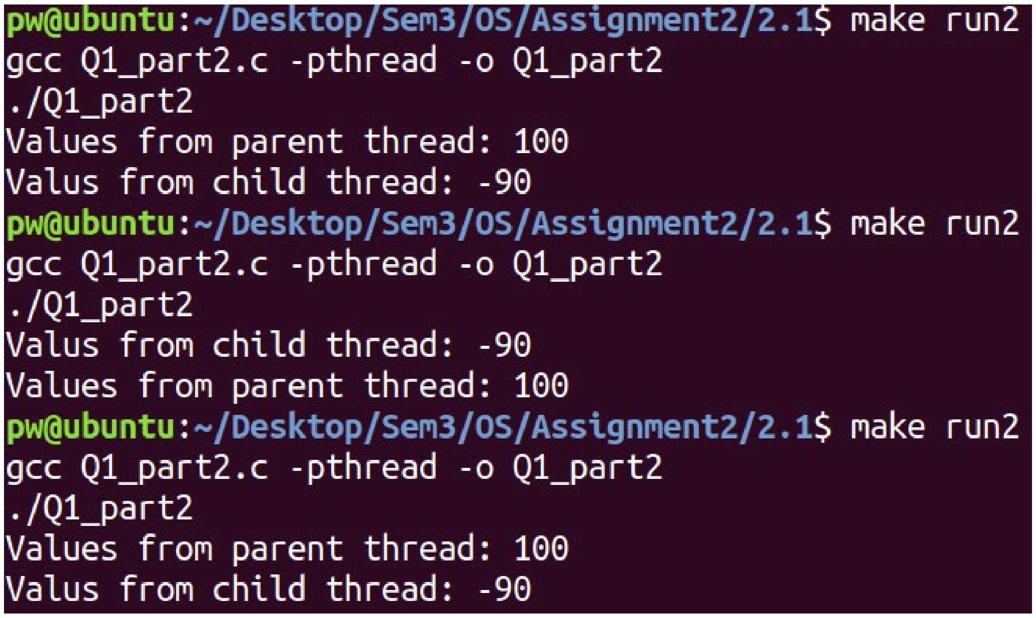


```
pw@ubuntu:~/Desktop/Sem3/OS/Assignment2/2.1$ make run1
gcc Q1_part1.c -o Q1_part1
./Q1_part1
Value from the parent process: 100
Value from the child process: -90
```

- The above output would be shown when the run command is executed .
- Here at first the parent process will run and then because of the waitpid function, it would wait for the child process to finish.
- Once the child process has exited, the parent process would finally terminate the execution of the program.

2 Thread

- It does not create a whole new process, rather it shares the memory space of the initial process with the child(newly created process).
- The new thread shares the memory and variables of the parent thread.
- Change in variables of any of the parent or child threads affects the variables of the other.
- Since both the threads share the memory space and variables, if any one of these changes the global variable, the change would get reflected in the other.



```
pw@ubuntu:~/Desktop/Sem3/OS/Assignment2/2.1$ make run2
gcc Q1_part2.c -pthread -o Q1_part2
./Q1_part2
Values from parent thread: 100
Valus from child thread: -90
pw@ubuntu:~/Desktop/Sem3/OS/Assignment2/2.1$ make run2
gcc Q1_part2.c -pthread -o Q1_part2
./Q1_part2
Valus from child thread: -90
Values from parent thread: 100
pw@ubuntu:~/Desktop/Sem3/OS/Assignment2/2.1$ make run2
gcc Q1_part2.c -pthread -o Q1_part2
./Q1_part2
Values from parent thread: 100
Valus from child thread: -90
```

- The above output would be shown when the run command is executed .
- As it is evident from the above output, since the variables are shared between the two threads, any of the child or parent process can terminate before the other.
- If the variable is printed at each iteration, it can be shown that there could be anomalies as the variable might be increasing, and before reaching the target, it starts to decrease and vice-versa.
- The above also proves that the threads are not waiting for any one of them to be over before the other is executed.