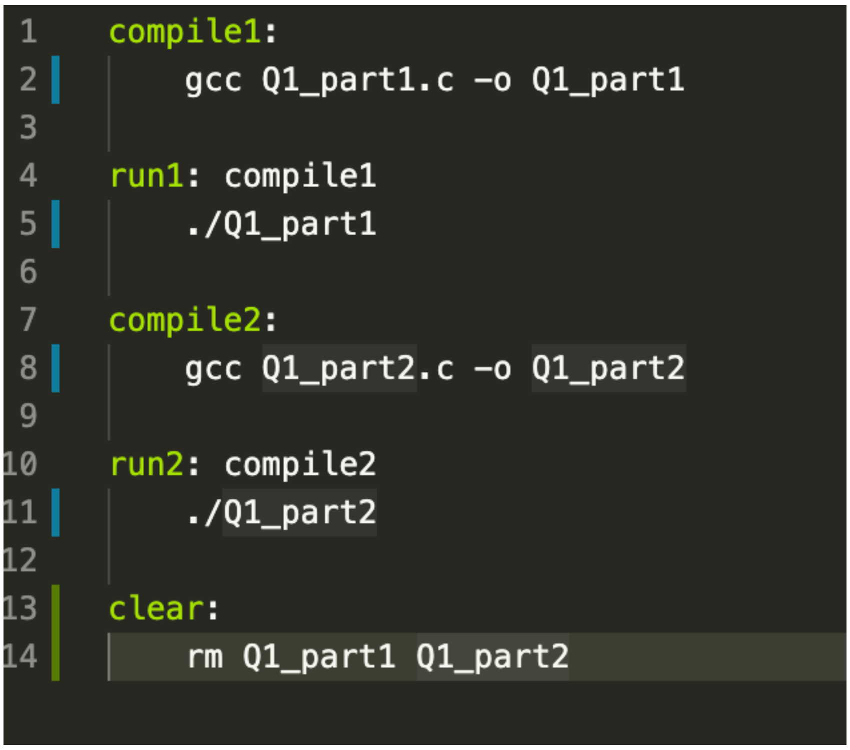
CSE231 – Operating Systems Assignment 2 – Q1 Writeup

Name : Ansh Arora
Roll no : 2019022

The makefile for the program –

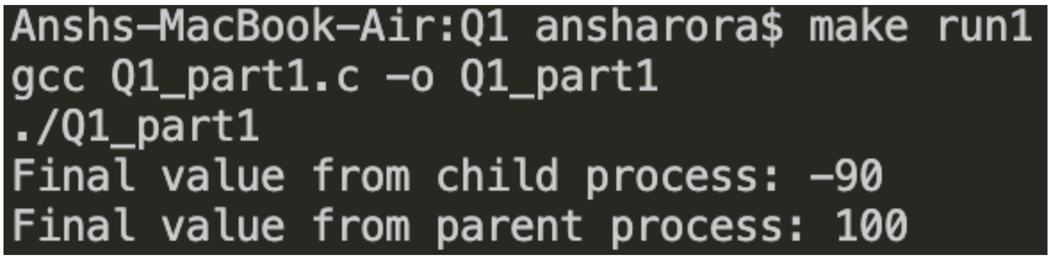


```
compile1:
	gcc Q1_part1.c -o Q1_part1

run1: compile1
	./Q1_part1

compile2:
	gcc Q1_part2.c -o Q1_part2

run2: compile2
	./Q1_part2

clear:
	rm Q1_part1 Q1_part2
```

1. Run make run1 for performing the process using the parent process/child process method (fork() and wait()).
2. Run make run2 for performing the process using the parent thread/child thread method (pthread_create() and pthread_join()).
3. Run make clear to reset the program.

Outputs observed –

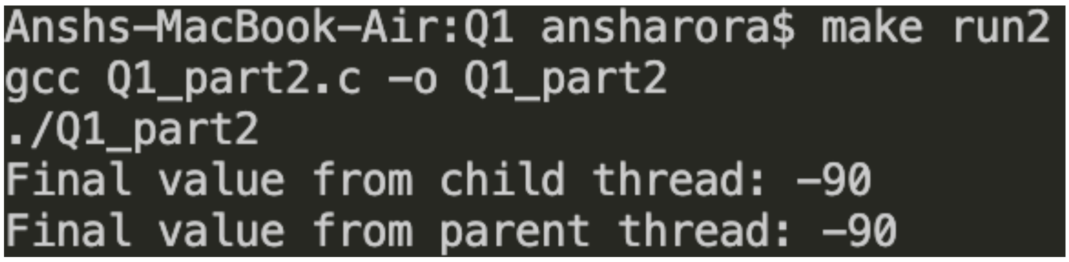
1. Using fork() (run1) –



```
Anshs-MacBook-Air:Q1 ansharora$ make run1
gcc Q1_part1.c -o Q1_part1
./Q1_part1
Final value from child process: -90
Final value from parent process: 100
```

Here the child process returns the value -90 and after it terminates and returns to the parent process, the parent process displays the final value 100. This is because when creating a process, the global variable data part of the memory does not get shared between the child and the parent process. As a result of this, both the child and the parent process have their own individual data memory, even for global variables and they both operate on their own individual data. So, the child process’s version of the global variable is made to reduce to -90 using a while loop, while on the other hand, the parent process’s version of global variable in its data section of the memory is increased to 100. So even after the child process terminates after having run concurrently with the parent process and returns to the parent process using the command wait(), the data of the parent process remains unaffected by the child process and displays 100.

2. Using pthread_create() (run2) –



```
Anshs-MacBook-Air:Q1 ansharora$ make run2
gcc Q1_part2.c -o Q1_part2
./Q1_part2
Final value from child thread: -90
Final value from parent thread: -90
```

Here both the child thread and parent threads return the same value i.e. -90. This occurs because of the fact that both the threads created by a process, even though might have a different code section, share the same data section of memory and hence have the same global variable locations to alter. When we create a child thread and alter its value down to -90, we concurrently use the parent thread to alter the same global variable to 100. As a result of this, when the child thread returns back to the parent thread using the method pthread_join(), the global variable of the parent gets altered to be same as the child thread, since they share memory, as a result of which the parent and child process both result in the final value of -90. The actual processing that goes behing the scenes differs in each run, since the two threads run concurrently and hence unpredictably.

Thus, we can say that the difference of ouputs arises due to the fact that global variables are shared between child and parent threads, but are different for child and parent processes.