EE4013 - C AND DATA STRUCTURES

EE18BTECH11017

G Yashwanth Naik

GATE2021-QP-CS-2, Q42

Problem

Consider the following multi-threaded code segment (in a mix of C and psuedo-code), invoked by two processes P1 and P2, and each of the processes spawns two threads T1 and T2:

```
int x = 0; // global
lock L1; // global
main(){
 create a thread to execute foo(); // Thread T1
 create a thread to execute foo(); // Thread T2
 wait for the two threads to finish execution;
 print(x);}
foo() {
 int y = 0;
 Acquire L1;
 x = x + 1;
 y = y + 1;
 Release L1;
 print(y);}
```

Problem

Which of the following statement(s) is/are correct?
(A) Both P1 and P2 will print the value of x as 2.
(B) At least one of P1 and P2 will print the value of x as 4.
(C) At least one of the threads will print the value of y as 2.
(D) Both T1 and T2, in both the processes will print the value of y as 1.
Both Options (A) and (D) are Correct.

Solution

For the Process P_1:

- Two threads are created in the main let they be T_{11} and T_{12}.
- Both the threads executes the foo() function and they don't wait for each other.Due to the locking mechanism in the foo() function, as soon as one of the threads enters the foo() it locks the function and doesn't allow other thread to execute simultaneously.
- Let T_{11} be the first thread to enter foo(),then it locks the function and increments **x to 1**, y being the local variable it prints **y as 1** and releases the lock.
- s soon as the T_{11} releases the lock, T_{12} executes foo() by locking, incrementing **x to 2** (since x is global variable), printing **y as 1**(local variable), unlocking.
- Finally, as soon as both the threads finishes their execution, **x** will be printed as **2**.

Solution

For the Process P_2:
The execution for P_2 will be same as P_1 because both the processes have different stack space, heap, text, data and also there is no part of the code which has mechanisms like shared memory,files etc.. so,both the processes will run independently.

Verifying Options

(A) Both P1 and P2 will print the value of x as 2.
True, because both the processes being independent have their own copy of variables so,they finally print x as 2.

(B) At least one of P1 and P2 will print the value of x as 4.
False, as both proccess print x as 2, none of them prints x as 4.

(C) At least one of the threads will print the value of y as 2.
False, since y is an local variable so both the threads have their own copy of variable y, which they would increment to 1 and print them as 1.

(D) Both T1 and T2, in both the processes will print the value of y as 1.
True, both threads will print x as 1 as mentioned above in (C) option.

Implementation

As the code-segment was invoked by 2 processes, the following C code invoke.c invokes the following C code(multithreading.c) with two processes(creating using fork()) via executable file, which represents the given code-segment with multiple threads

Output

The code invoke.c(creating multiple processes) produces the following output:

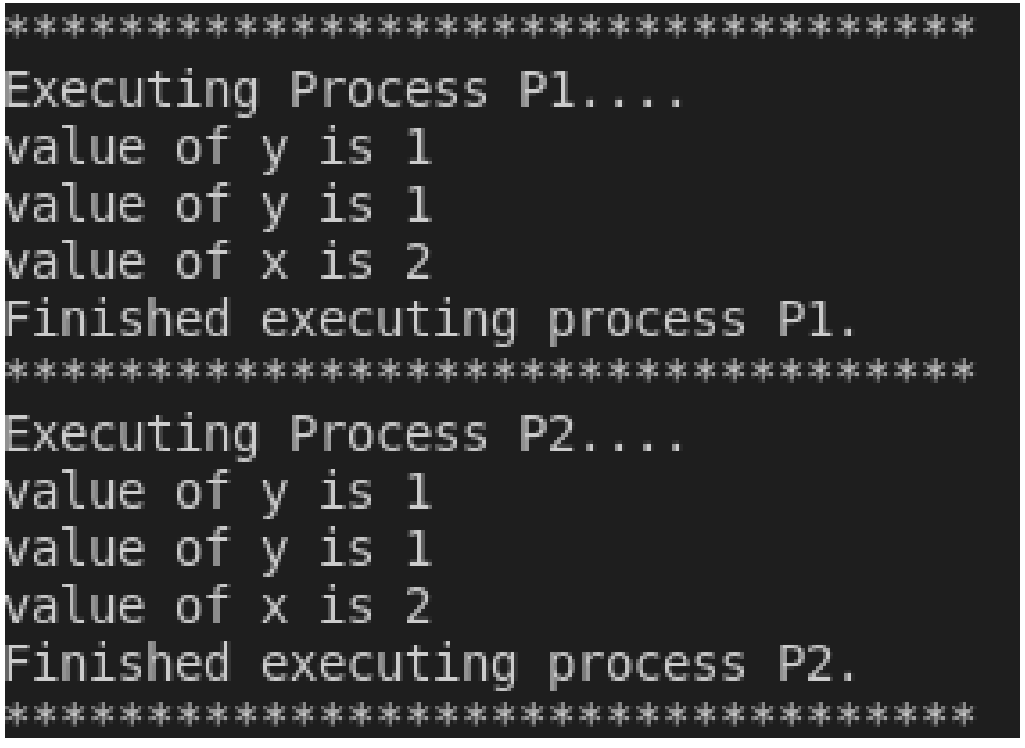

Figure 1: Output

Proof of Multithreading:

To demonstrate multithreading, two differennt functions and threads are created.Those two functions are called using these two threads.As multithreading occurs here we must observe the overlap of execution of these functions. The following code (multithreadproof.c) prints the following output.

```
********** Thread1 execution started **********
********** Thread2 execution started **********
Time elapsed from start of program to enter foo1 of thread2 311 us
Time elapsed from start of program to end of foo1 of thread2 332 us
********** Thread2 execution ended **********
Time elapsed from start of program to enter foo of thread1 284 us
Time elapsed from start of program to end of foo of thread1 370 us
********** Thread1 execution ended **********
value of x is 2
```

Figure 2: Output1

We can observe that thread1 execution overlaps with thread2 execution.