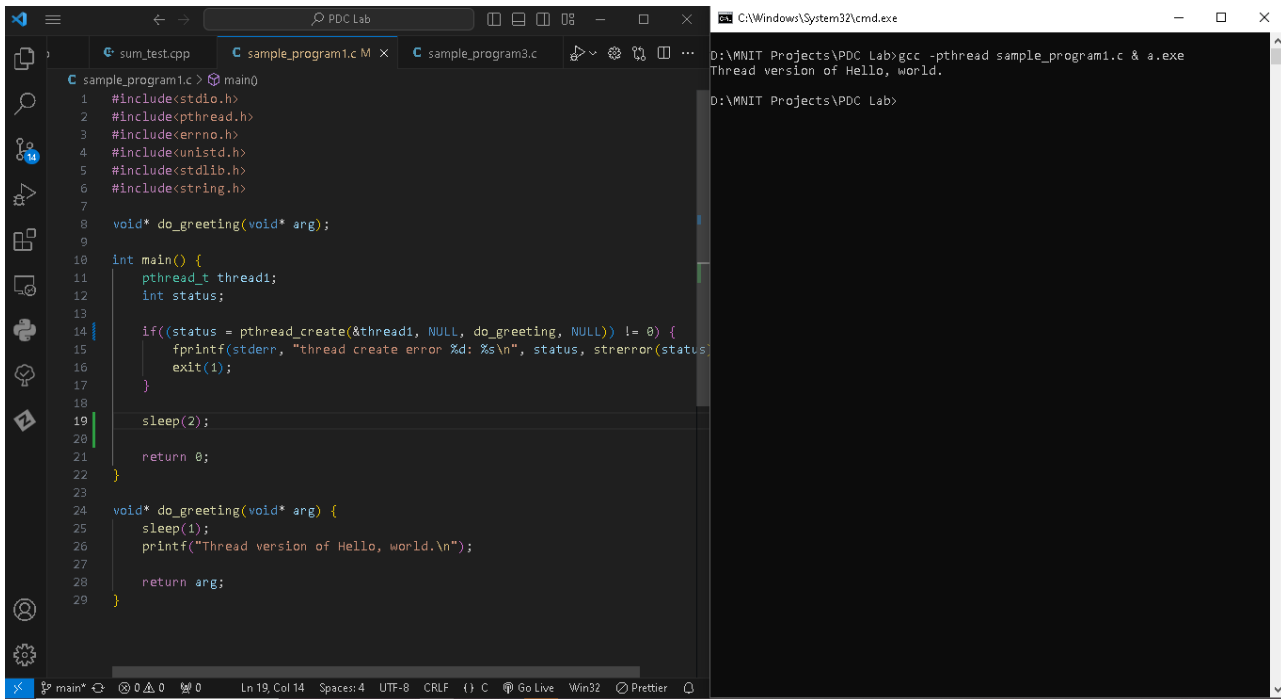


**Name: Ranjan Baro**

**ID: 2023PCP5274**

1. Describe/explain your observations, i.e. what must have happened in the original, unmodified Sample 1 program?

**Ans.** When original program is run, we do not observe any output or print statement. After inserting a **2 seconds *sleep* ( )** into the ***main*** function (after thread creation), we can observe the following output.

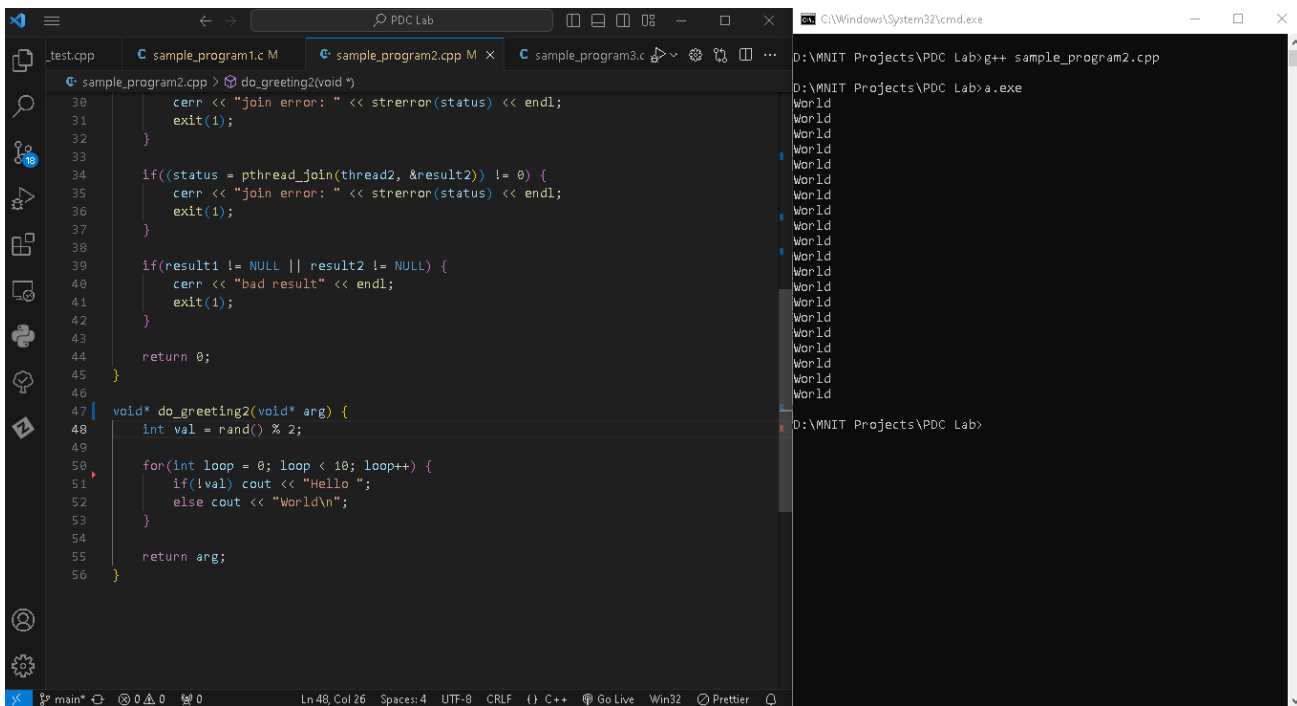


The screenshot shows a code editor with a C program named `sample_program1.c`. The program includes `stdio.h`, `pthread.h`, `errno.h`, `unistd.h`, `stdlib.h`, and `string.h`. It defines a function `do_greeting` that prints "Thread version of Hello, world." and sleeps for 1 second. The `main` function creates a thread `thread1` that calls `do_greeting`, then sleeps for 2 seconds before returning. The terminal output shows the command `gcc -pthread sample_program1.c & a.exe` and the output `Thread version of Hello, world.`.

We have not observed any output in original program, because the main thread and the created thread run concurrently and the program may exit before the created thread complete its execution of ***do\_greeting*** function. When we insert **2 seconds *sleep*** in the ***main*** function, the main program wait for 2 seconds before exiting and during this wait, the created thread will complete its execution of ***do\_greeting*** function and show the print statement of the ***do\_greeting*** function.

2. Report your results, particularly the observed formatting of the sample 2 program.

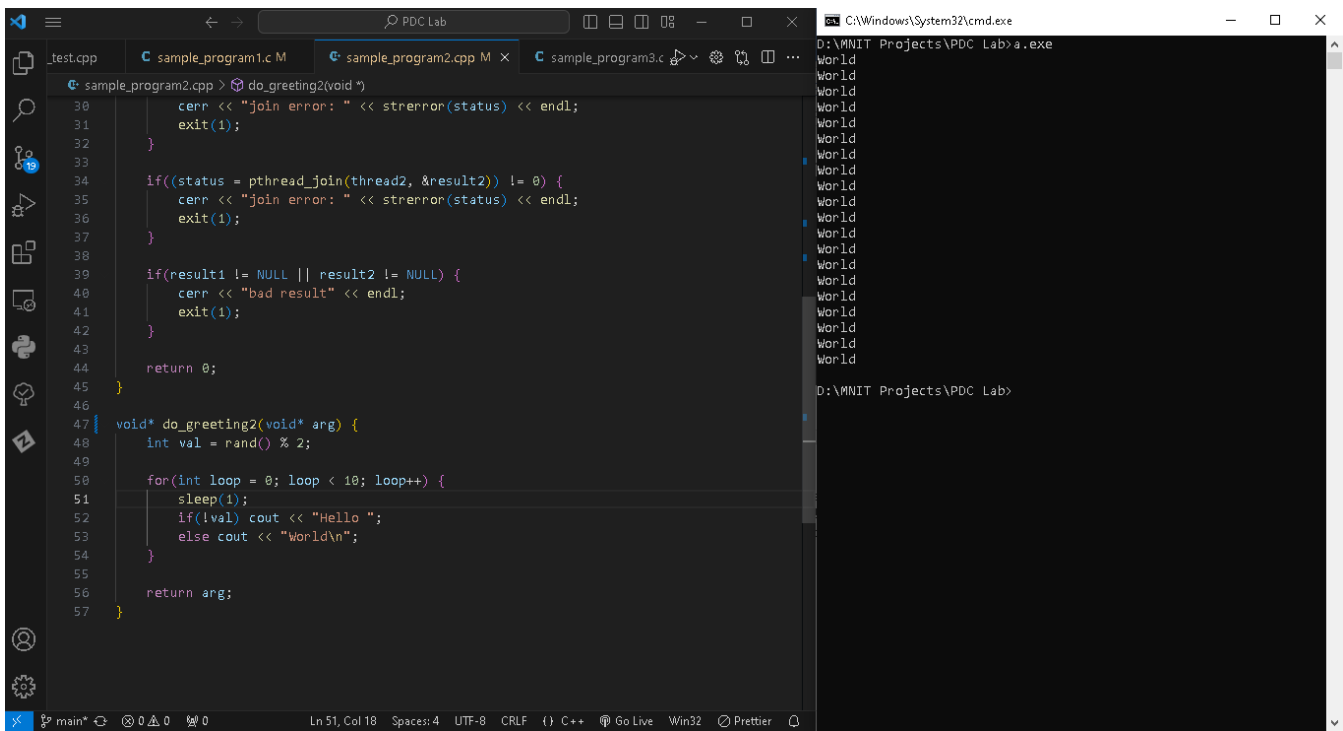
**Ans.** Sample program 2 generates two thread and both thread execute ***do\_greeting2*** function. The two thread run concurrently and may print **Hello** or **World** depending on the generated random variable **val**. We may observe the following output.



```
sample_program2.cpp > do_greeting2(void *)
30     cerr << "join error: " << strerror(status) << endl;
31     exit(1);
32 }
33
34 if((status = pthread_join(thread2, &result2)) != 0) {
35     cerr << "join error: " << strerror(status) << endl;
36     exit(1);
37 }
38
39 if(result1 != NULL || result2 != NULL) {
40     cerr << "bad result" << endl;
41     exit(1);
42 }
43
44 return 0;
45 }
46
47 void* do_greeting2(void* arg) {
48     int val = rand() % 2;
49
50     for(int loop = 0; loop < 10; loop++) {
51         if(!val) cout << "Hello ";
52         else cout << "World\n";
53     }
54
55     return arg;
56 }
```

```
D:\MNIT Projects\POC Lab>g++ sample_program2.cpp
D:\MNIT Projects\POC Lab>a.exe
World
World
World
World
World
World
World
World
World
World
D:\MNIT Projects\POC Lab>
```

3. Report your results again. Explain why they are different from the results seen in question 2.  
**Ans.** When we insert **one second sleep ( )** at the beginning of the loop in the **do\_greeting2 ( )** function, we see delay in the execution of threads. There is one second delay between each print statement **Hello** or **World**. Here is the output.



```
sample_program2.cpp > do_greeting2(void *)
30     cerr << "join error: " << strerror(status) << endl;
31     exit(1);
32 }
33
34 if((status = pthread_join(thread2, &result2)) != 0) {
35     cerr << "join error: " << strerror(status) << endl;
36     exit(1);
37 }
38
39 if(result1 != NULL || result2 != NULL) {
40     cerr << "bad result" << endl;
41     exit(1);
42 }
43
44 return 0;
45 }
46
47 void* do_greeting2(void* arg) {
48     int val = rand() % 2;
49
50     for(int loop = 0; loop < 10; loop++) {
51         sleep(1);
52         if(!val) cout << "Hello ";
53         else cout << "World\n";
54     }
55
56     return arg;
57 }
```

```
D:\MNIT Projects\POC Lab>a.exe
World
World
World
World
World
World
World
World
World
World
D:\MNIT Projects\POC Lab>
```

5. Compile the sample program 3 and run it multiple times (you may see some variation between runs).  
Choose one particular sample run. Describe, trace, and explain the output of the program.  
**Ans.** In sample 3 program, the **main** thread creates two additional threads. The main thread print **sharedData** and other two thread also print **sharedData** along with char **val (a or b)**. Here is the output of the program.

The screenshot shows a code editor with three tabs: `sample_program1.c M`, `sample_program2.cpp M`, and `sample_program3.c M`. The active tab is `sample_program3.c`, which contains the following C code:

```
33 void* do_greeting3(void*) {
34     if((status = pthread_join(thread2, &result2)) != 0) {
35         fprintf(stderr, "join error %d: %s\n", status, strerror(status));
36         exit(1);
37     }
38     if((status = pthread_join(thread2, &result2)) != 0) {
39         fprintf(stderr, "join error %d: %s\n", status, strerror(status));
40         exit(1);
41     }
42     printf("Parent sees %d\n", sharedData);
43     return 0;
44 }
45
46 void* do_greeting3(void* arg) {
47     char *val_ptr = (char *)arg;
48     printf("Child receiving %c initially sees %d\n", *val_ptr, sharedData);
49     sleep(1);
50     sharedData++;
51     printf("Child receiving %c now sees %d\n", *val_ptr, sharedData);
52     return NULL;
53 }
```

The terminal window on the right shows the execution of the program. It runs `gcc -pthread sample_program3.c` and then `./a.exe`. The output shows the parent thread printing "Parent sees 5" and then the two child threads printing their initial and updated values of `sharedData` (5 and 6 respectively).

Execution steps of the program -

- i) Main thread creates two threads which execute `do_greeting3` function passing the character value **a** or **b**. After creating two threads, it prints `sharedData`'s value i.e. **Parent sees 5**.
- ii) The two child thread start executing `do_greeting3` function. Each child thread then prints `sharedData`.
- iii) After 1 second sleep, each child updates `sharedData`'s value and prints the `sharedData`'s value.
- iv) At last the main thread print the updated value of `sharedData`.

6. Explain in your own words how the thread-specific (not shared) data is communicated to the child threads.

**Ans.** Each thread in a multi thread program have a local copy of a variable. This variable can be updated without interfering with other thread's variable. Main thread can set and get values for this variable. This allows it to communicate initial values or any updates to the child thread.