Name : Ramzy Izza Wardhana
NIM : 21/472698/PA/20322
Class : IUP CS B

Activity 7.2 Lab Computer Systems & Networking

Activity 7.2

Modify the second thread function to have the same order of accessing the resources as the first thread function.

- What is the output of the program?
- What happens?

Step 1: Change the second function to have the same order of accessing the resource as the first function

```
void *function1() {
        pthread_mutex_lock(&res_a);
        printf("Thread ONE acquired res_a\n");
        sleep(1);

        pthread_mutex_lock(&res_b);
        printf("Thread ONE acquired res_b\n");

        pthread_mutex_unlock(&res_b);
        printf("Thread ONE released res_b\n");

        pthread_mutex_unlock(&res_a);
        printf("Thread ONE released res_a\n");
}

void *function2(){
        pthread_mutex_lock(&res_a);
        printf("Thread TWO acquired res_b\n");
        sleep(1);

        pthread_mutex_lock(&res_b);
        printf("Thread TWO acquired res_a\n");

        pthread_mutex_unlock(&res_b);
        printf("Thread TWO released res_a\n");

        pthread_mutex_unlock(&res_a);
        printf("Thread TWO released res_b\n");
}
```

Observe that on second function, the parameter of each thread (pthread_mutex_lock and pthread_mutex unlock) has the same order as the first function.

Step 2: Compile the program

```
root@--:~# nano deadlock.c
root@--:~# gcc -o deadlock.out deadlock.c
```

Step 3: Run the Program

```
root@--:~# ./deadlock.out
Thread ONE acquired res_a
Thread ONE acquired res_b
Thread ONE released res_b
Thread ONE released res_a
Thread TWO acquired res_b
Thread TWO acquired res_a
Thread TWO released res_a
Thread TWO released res_b
Thread joined
```

The underlying process behind the output besides illustrates that deadlock does not occur and both functions had successfully completed their process. The reason is because in this case, when both functions run in a same time quantum, function1 and function2 need to acquire a. Since function1 runs first before function2, thus making function1

locked a first, which then function2 needs to wait for a until the process of function1 finished. Next, function1 will continues to execute to acquire b and will release both a and b. After waiting for funcion1 to complete its task, function2 can proceed to acquire a and b because both of them are already released previously by function1. Hence from the explanation above, it can be summarized that deadlock does not happen that was caused by function1 will release to function2 once its finished and function1 and function2 doesn't need to wait indefinitely.