OPERATING SYSTEM – Lab 5

Submitted by : Ayesha Javaid

Roll # : BITF22M523

Deadlock Detection and Avoidance

1. Is Deadlock Possible?

Yes, **deadlock is possible** in this program.

- Each thread (`thread1`, `thread2`, `thread3`) locks resources (`m1`, `m2`, `m3`) in **different orders**.
- If one thread acquires a lock and another thread acquires a different lock, they can **block each other** indefinitely while waiting for the other lock to be released.

2. Resource Allocation Graph (RAG)

- **Threads (T1, T2, T3)** represent processes.
- **Resources (m1, m2, m3)** represent mutex locks.

Graph Representation:

- **T1 → m1 → T1 → m2 → T1 → m3**
- **T2 → m2 → T2 → m3 → T2 → m1**
- **T3 → m3 → T3 → m1 → T3 → m2**

Cycle Detected:

- There is a **circular wait condition** because each thread holds one mutex while waiting for the next mutex held by another thread.

Conclusion: Deadlock **exists**.

3. Deadlock Conditions (Coffman Conditions)

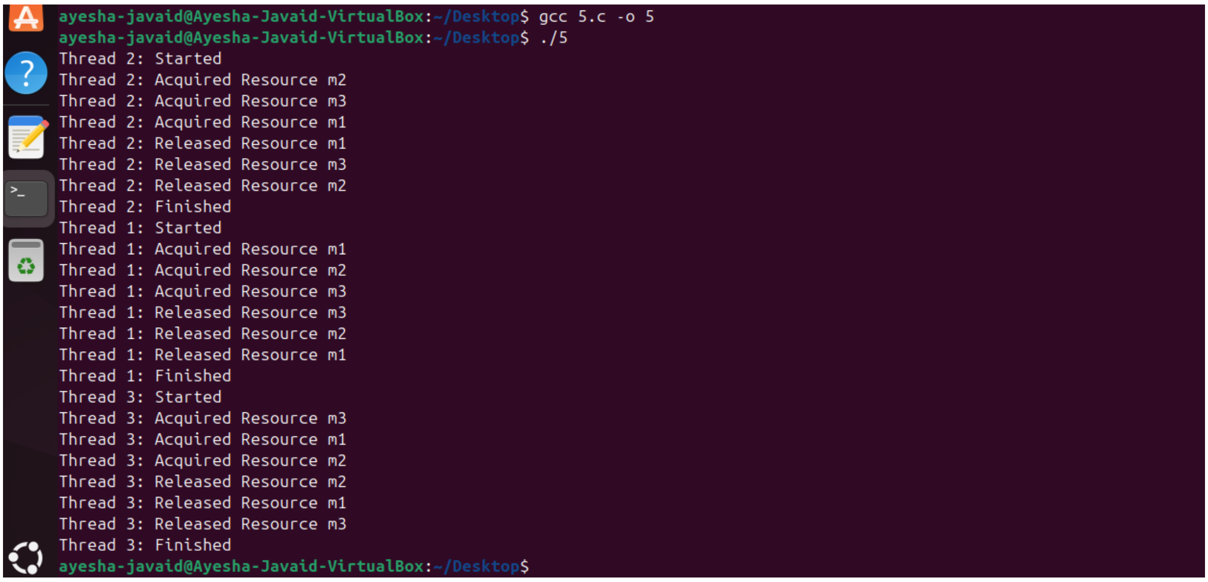
1. **Mutual Exclusion:** Each resource (`m1`, `m2`, `m3`) can only be held by one thread at a time. ✅ Satisfied
2. **Hold and Wait:** Threads hold some resources while waiting for others. ✅ Satisfied
3. **No Preemption:** Resources cannot be forcibly taken from threads. ✅ Satisfied
4. **Circular Wait:** Threads form a circular chain, where each thread waits for a resource held by the next thread. ✅ Satisfied

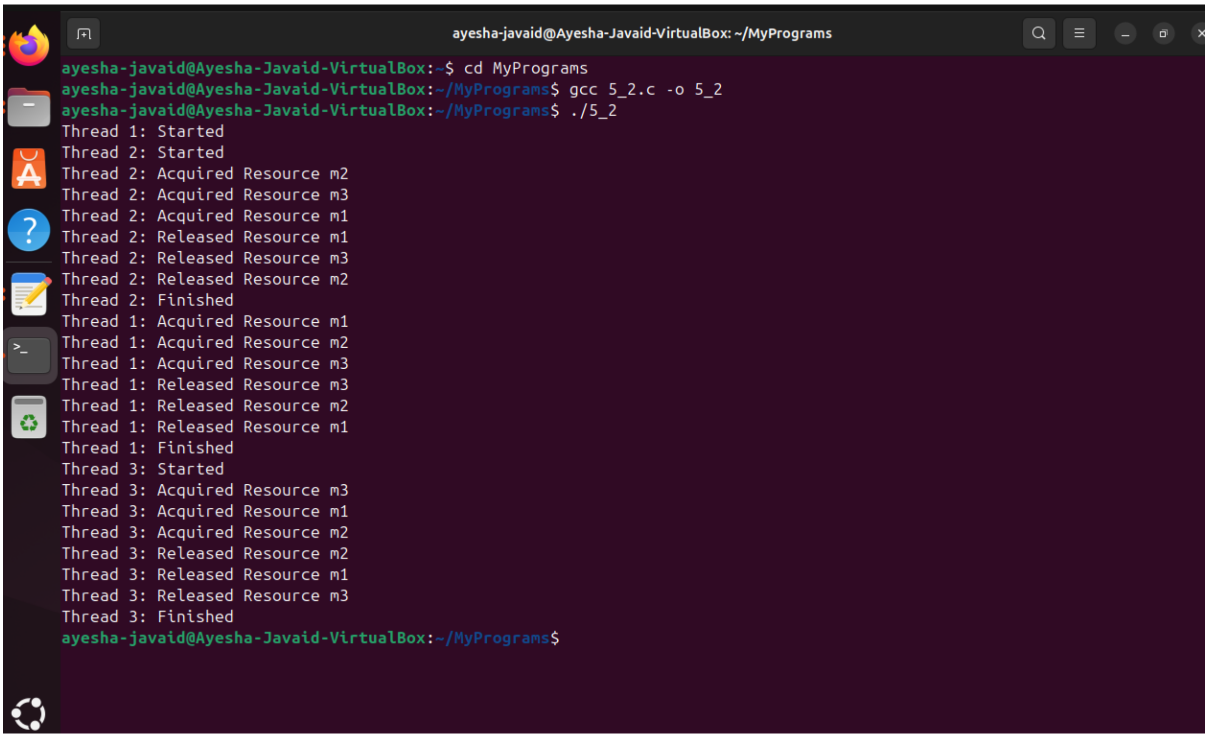
Code Reference Example:

- `thread1` locks `m1` → waits for `m2` → waits for `m3`
- `thread2` locks `m2` → waits for `m3` → waits for `m1`
- `thread3` locks `m3` → waits for `m1` → waits for `m2`

4. Deadlock Avoidance Using Banker's Algorithm

Banker's Algorithm can be applied to ensure **safe resource allocation sequences** and prevent deadlock.


ayesha-javaid@Ayesha-Javaid-VirtualBox:~/Desktop$ gcc 5.c -o 5
ayesha-javaid@Ayesha-Javaid-VirtualBox:~/Desktop$ ./5
Thread 2: Started
Thread 2: Acquired Resource m2
Thread 2: Acquired Resource m3
Thread 2: Acquired Resource m1
Thread 2: Released Resource m1
Thread 2: Released Resource m3
Thread 2: Released Resource m2
Thread 2: Finished
Thread 1: Started
Thread 1: Acquired Resource m1
Thread 1: Acquired Resource m2
Thread 1: Acquired Resource m3
Thread 1: Released Resource m3
Thread 1: Released Resource m2
Thread 1: Released Resource m1
Thread 1: Finished
Thread 3: Started
Thread 3: Acquired Resource m3
Thread 3: Acquired Resource m1
Thread 3: Acquired Resource m2
Thread 3: Released Resource m2
Thread 3: Released Resource m1
Thread 3: Released Resource m3
Thread 3: Finished
ayesha-javaid@Ayesha-Javaid-VirtualBox:~/Desktop$


ayesha-javaid@Ayesha-Javaid-VirtualBox: ~/MyPrograms
ayesha-javaid@Ayesha-Javaid-VirtualBox:~$ cd MyPrograms
ayesha-javaid@Ayesha-Javaid-VirtualBox:~/MyPrograms$ gcc 5_2.c -o 5_2
ayesha-javaid@Ayesha-Javaid-VirtualBox:~/MyPrograms$ ./5_2
Thread 1: Started
Thread 2: Started
Thread 2: Acquired Resource m2
Thread 2: Acquired Resource m3
Thread 2: Acquired Resource m1
Thread 2: Released Resource m1
Thread 2: Released Resource m3
Thread 2: Released Resource m2
Thread 2: Finished
Thread 1: Acquired Resource m1
Thread 1: Acquired Resource m2
Thread 1: Acquired Resource m3
Thread 1: Released Resource m3
Thread 1: Released Resource m2
Thread 1: Released Resource m1
Thread 1: Finished
Thread 3: Started
Thread 3: Acquired Resource m3
Thread 3: Acquired Resource m1
Thread 3: Acquired Resource m2
Thread 3: Released Resource m2
Thread 3: Released Resource m1
Thread 3: Released Resource m3
Thread 3: Finished
ayesha-javaid@Ayesha-Javaid-VirtualBox:~/MyPrograms$