Operating Systems

Assignment 03

Title: Deadlock handling methods & Prevention

Compiled By: Alina Liaquat

Roll-No: BSCSM-A-23-19

BSCS-MORNING-5th(A)

Submitted to: Sir Umar Malik

UNIVERSITY OF LAYYAH MAIN CAMPUS HAFIZABAD

DEADLOCK HANDLING METHODS

1. OBJECTIVE

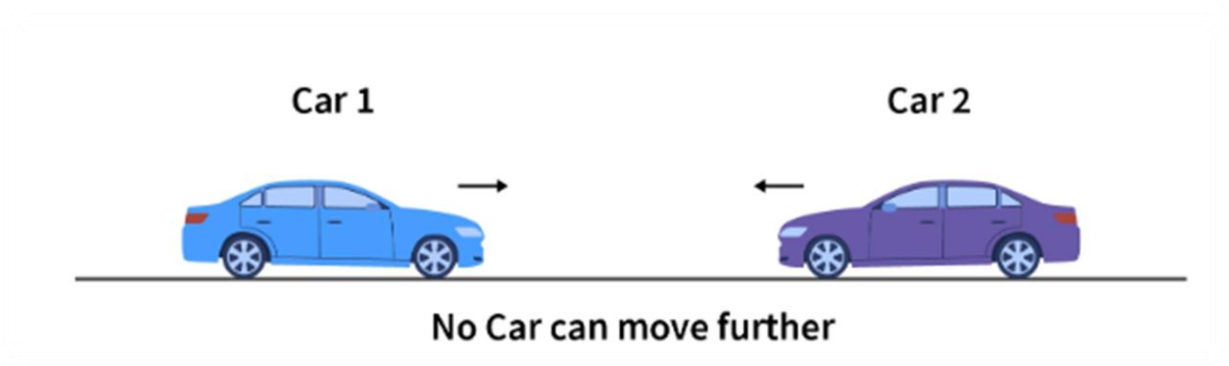
The objective of this practical assignment is to understand:

- What deadlock is
- How synchronization mechanisms (mutex, semaphore) control concurrent access
- Methods to **prevent deadlocks**
- Implementing a C++ program that **causes a deadlock** and another that **prevents** it

2. WHAT IS DEADLOCK?

Deadlock occurs when two or more processes/threads are permanently blocked, waiting for resources held by each other.

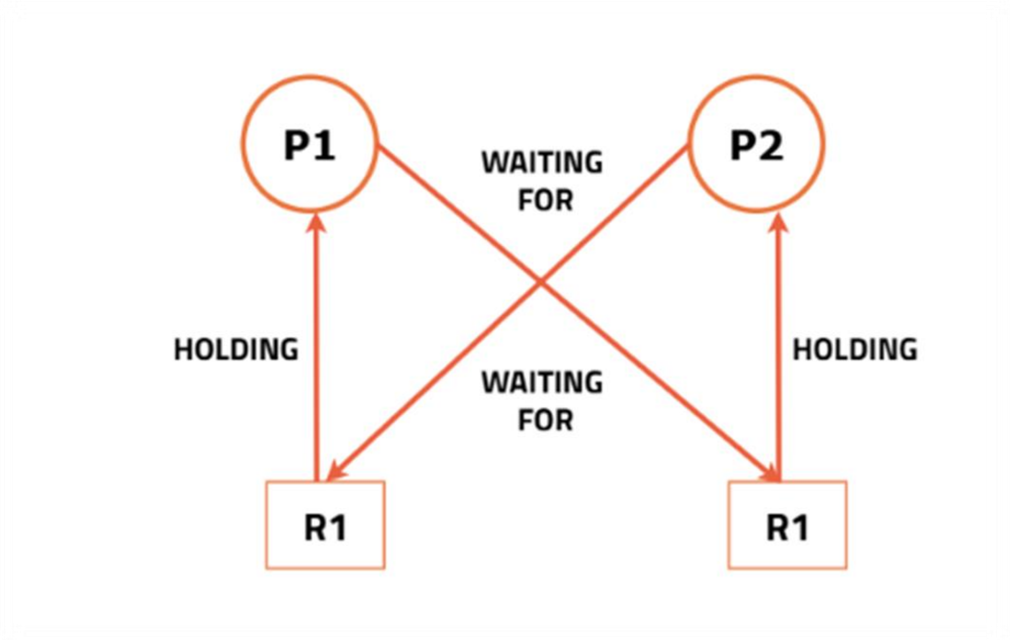
Consider a one-way road with two cars approaching from opposite directions, blocking each other. The road is the resource and crossing it represents a process. Since it’s a one-way road, both cars can’t move simultaneously, leading to a deadlock.



3. CONDITIONS FOR DEADLOCK

Deadlock happens only when all 4 conditions are true:

1. Mutual Exclusion
2. Hold and Wait
3. No Preemption
4. Circular Wait

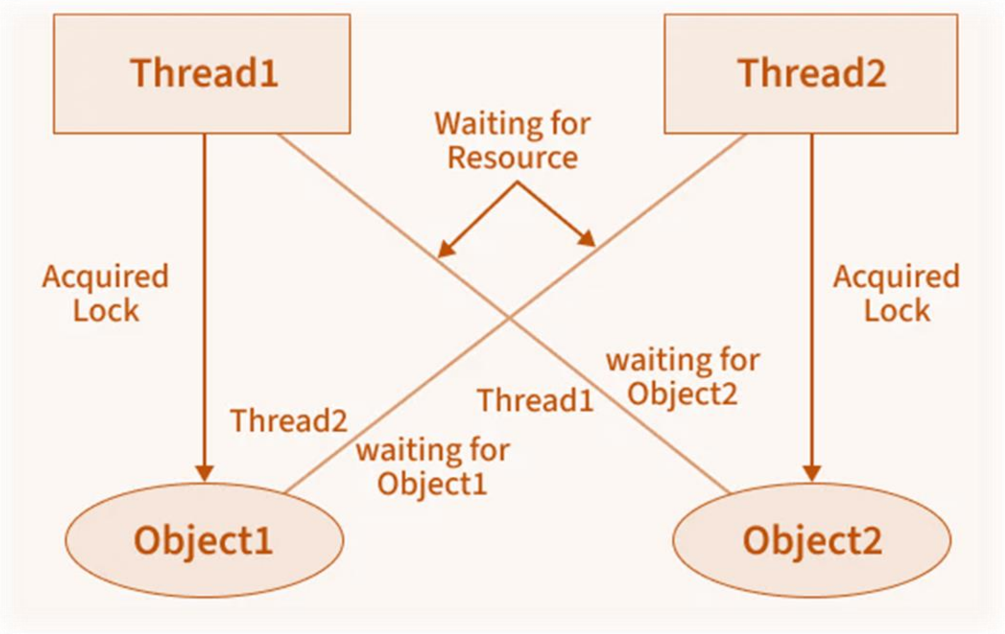


4. DEADLOCK DEMONSTRATION (C++ Program)

We first create a deadlock example using **two mutex locks**

4.1 LOGIC

- Thread 1 locks resourceA then waits for resourceB
- Thread 2 locks resourceB then waits for resourceA
- Both wait forever → **deadlock**



4.2 C++ CODE (DEADLOCK EXAMPLE)

```
#include <iostream>

#include <thread>

#include <mutex>

using namespace std;


mutex resourceA;

mutex resourceB;


void task1() {

    lock_guard<mutex> lockA(resourceA);

    cout << "Task 1 locked Resource A\n";


    this_thread::sleep_for(chrono::milliseconds(100));


    cout << "Task 1 waiting for Resource B...\n";
```

```
    lock_guard<mutex> lockB(resourceB); // waits forever
    cout << "Task 1 got Resource B\n";
}

void task2() {
    lock_guard<mutex> lockB(resourceB);
    cout << "Task 2 locked Resource B\n";

    this_thread::sleep_for(chrono::milliseconds(100));

    cout << "Task 2 waiting for Resource A...\n";
    lock_guard<mutex> lockA(resourceA); // waits forever
    cout << "Task 2 got Resource A\n";
}

int main() {
    thread t1(task1);
    thread t2(task2);

    t1.join();
    t2.join();

    return 0;
}
```

Output
Task 1 locked Resource A Task 2 locked Resource B Task 1 waiting for Resource B... Task 2 waiting for Resource A... (Hangs forever – deadlock)

5. DEADLOCK PREVENTION

We fix the problem by using a global resource ordering rule:

All threads must lock resources in the same order
Example: Always lock A → then B

This breaks Circular Wait, preventing deadlock

5.1 C++ CODE PREVENTING DEADLOCK

```
#include <iostream>

#include <thread>

#include <mutex>

using namespace std;



mutex resourceA;

mutex resourceB;



// Both functions lock A → B order

void safeTask1() {

    lock_guard<mutex> lockA(resourceA);
```

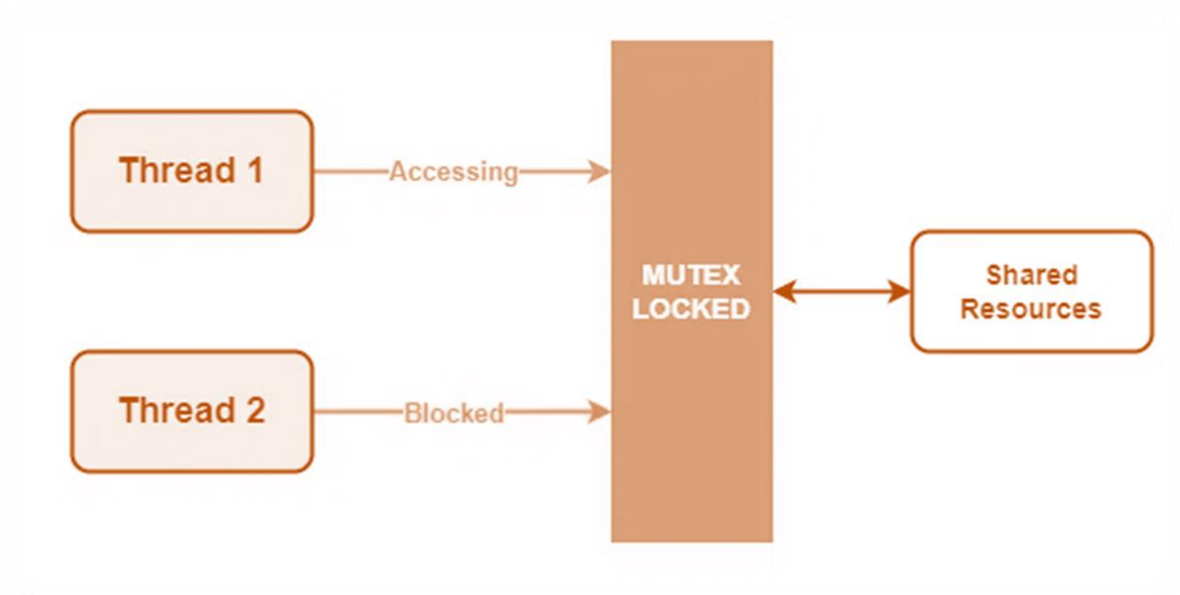
```
    cout << "Task 1 locked Resource A\n";

    this_thread::sleep_for(chrono::milliseconds(100));

    lock_guard<mutex> lockB(resourceB);
    cout << "Task 1 locked Resource B\n";
}

void safeTask2() {
    lock_guard<mutex> lockA(resourceA);
    cout << "Task 2 locked Resource A\n";

    this_thread::sleep_for(chrono::milliseconds(100));

    lock_guard<mutex> lockB(resourceB);
    cout << "Task 2 locked Resource B\n";
}

int main() {
    thread t1(safeTask1);
    thread t2(safeTask2);

    t1.join();
    t2.join();
    return 0;
}
```

Output
Task 1 locked Resource A Task 1 locked Resource B Task 2 locked Resource A Task 2 locked Resource B (No deadlock)

6. SYNCHRONIZATION TECHNIQUES USED

6.1 MUTEX

- Ensures only one thread enters a critical section
  Use case: resource protection



7. DEADLOCK HANDLING METHODS SUMMARY

Method	Explanation
Deadlock Prevention	Break one of 4 conditions (we break circular wait)
Deadlock Avoidance	Banker’s Algorithm (not implemented here)
Deadlock Detection	Allow deadlock → detect → recover
Synchronization	Mutex/semaphore prevents race conditions

CONCLUSION

In this assignment, I have:

- Observed how deadlock occurs
- Implemented a real deadlock in C++
- Prevented it using proper synchronization
- Understood mutex usage and ordering rules

This demonstrates the fundamental concept of process synchronization and deadlock prevention in operating systems