P2 LFG SYNC

Presented By : Jaycee Montenegro

STDISCM S11 | 2025

Github Link:
https://github.com/Xiao-Alatus/STDISCM-LFG-Synchronization-Montenegro

POSSIBLE DEADLOCK SCENARIOS

Shutdown edge case:

If shutdown is triggered before all parties are served, some threads may remain blocked.

SYNCHRONIZATION MECHANISMS USED

```
//Gives all the instances time to finish
Thread.Sleep((int)(t2 + 1) * 1000);

Program.shutdown = true;
foreach (var instance in instances)
{
    instance.Signal.Set();
}

foreach (var instance in instances)
{
    instance.Join();
}
```

- **Used a sleep to make sure every instance is finished before shutting down all the threads.**

Since no locks are used and the dispatcher (main thread) directly assigns work to instance threads without cyclical waiting, deadlocks should not occur.

Are deadlocks in the code possible? lets check using the Coffman condition

Condition	Explanation	My Code
1.Mutual Exclusion	Threads hold exclusive access to resources	❌ No shared locks or exclusive resources
2.Hold and Wait	Threads hold one resource while waiting for another	❌ Threads don't hold anything while waiting
3.No Preemption	Resources cannot be forcibly taken; they must be released voluntarily.	✅ Technically True but safe, because nothing to preempt
4.Circular Wait	A closed chain of threads exists where each thread waits for a resource held by the next.	❌ No cycles, all threads wait on independent events

My code violates 3 of the 4 conditions, meaning a deadlock cannot occur.

POSSIBLE STARVATION SCENARIOS

Busy instances dominate usage:

Without proper scheduling, some instances may always get skipped if others finish faster.

SYNCHRONIZATION MECHANISMS USED

```
//Round Robin Party Dispatcher
uint currentIndex = 0;
while (maxNumberOfParties > 0)
{
    if (instanceStatus[currentIndex] == "empty")
    {
        instances[(int)currentIndex].AssignWork();
        maxNumberOfParties--;
    }

    currentIndex = (currentIndex + 1) % (uint)instances.Count;
    Thread.Sleep(50);
}
```

- **Implemented Round Robin scheduling to ensure fair and even work distribution, preventing starvation.**

Round Robin ensures that even if all instances have a finish time of 1 second, the work is still distributed fairly among them.

THANK YOU

Presented By : Jaycee Montengro