**Report**

Simulation works by creating a deadlock detector that is used to detect deadlocks. Then we use a Semaphore to manage the resources and threads. We have a resource class that is used to simulate the resources, and it will handle acquiring threads and releasing them. Finally, we have the process themselves where we can initiate different processes to simulate the whole thing in the main section.

In the main section I changed the order of join to prevent deadlocks, however this does not simulate proper multiprocessing, so I instead changed the deadlock resolver to properly remove deadlocks since initially it was not removing deadlocks and ran indefinitely.

The output observed was a deadlock running indefinitely and some print inconsistencies that I tried to resolve with a global mutex for console outputs; but I only solved so much of it.

I learned that the hardest part is resolving deadlocks and handling deadlocks. Mutexes are tricky to work with and blocking code alongside mutexes creates some bugs. Memory management is essential to resolving synchronization and recovering from deadlocks.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.