Tazhibayev Sultanbay

SE-2326

Assignment 6

Advanced Operating Systems | Seilkhanova Kymbat

Labwork 6: Investigating Multiprocessor Systems in Operating Systems Using Python

Goal:

The aim of this lab is to deepen your understanding of multiprocessor system architectures, especially shared-memory models, and how they are managed by modern operating systems.

Prerequisites:

A Python setup (preferably version 3.9 or newer)

Lab Outline:

1. Overview of Multiprocessor Architectures (Theory Section)

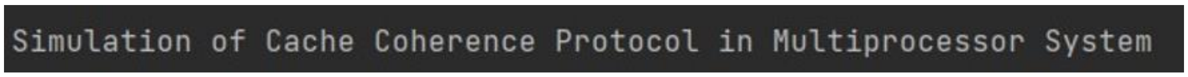
Goal:

Grasp the fundamental concepts of shared-memory multiprocessor architectures and their variations.

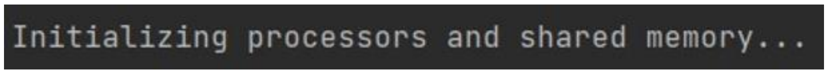
Instructions:

Study Chapter 8, which contains a detailed explanation of multiprocessor systems.

Compare and contrast Uniform Memory Access (UMA) with Non-Uniform Memory Access (NUMA) architectures. Write down the key differences.



```
Simulation of Cache Coherence Protocol in Multiprocessor System
```



```
Initializing processors and shared memory...
```

2. Simulating a Cache Coherence Protocol (Practical Section)

Cache coherence protocols help ensure data consistency across the individual caches of multiple processors that all access the same shared memory.

Think of it like this:

If several team members (processors) are working on the same document but each one keeps their own copy (cache), any updates made by one member need to be reflected in everyone else's copy to avoid mistakes. That's the job of a cache coherence protocol — it synchronizes updates so all processors see the same data.

Goal:

Build a simple simulation that demonstrates how a cache coherence mechanism works in a multiprocessor setup.

Instructions:

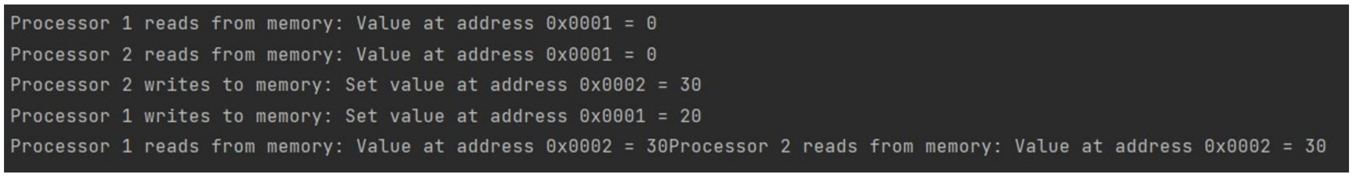
Create a Python script named `cache_coherence_simulation.py`.

Define a `SharedMemory` class to represent the central memory all processors interact with.

Create a `Processor` class to model individual threads/processors.

Simulate read and write operations from multiple processors and show how cache consistency is maintained.

Run your simulation and examine the output.



```
Processor 1 reads from memory: Value at address 0x0001 = 0
Processor 2 reads from memory: Value at address 0x0001 = 0
Processor 2 writes to memory: Set value at address 0x0002 = 30
Processor 1 writes to memory: Set value at address 0x0001 = 20
Processor 1 reads from memory: Value at address 0x0002 = 30Processor 2 reads from memory: Value at address 0x0002 = 30
```

```
Simulation completed.
```

3. Learning About Synchronization Techniques (Theory Section)

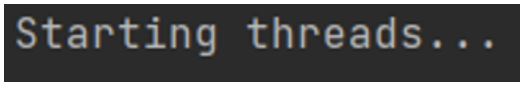
Goal:

Understand how synchronization works in systems with multiple processors.

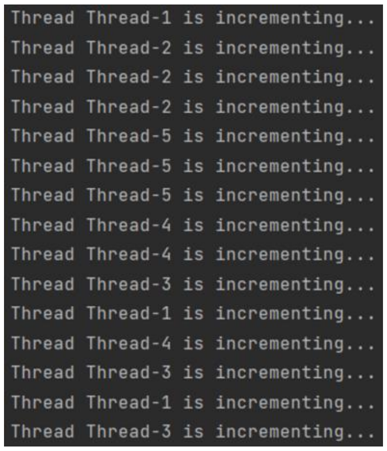
Instructions:

Read the theoretical background on synchronization mechanisms in Chapter 8.

Write a short explanation on the importance of mutexes in avoiding race conditions when multiple processors access shared data.



```
Starting threads...
```



```
Thread Thread-1 is incrementing...
Thread Thread-2 is incrementing...
Thread Thread-2 is incrementing...
Thread Thread-2 is incrementing...
Thread Thread-5 is incrementing...
Thread Thread-5 is incrementing...
Thread Thread-5 is incrementing...
Thread Thread-4 is incrementing...
Thread Thread-4 is incrementing...
Thread Thread-3 is incrementing...
Thread Thread-1 is incrementing...
Thread Thread-4 is incrementing...
Thread Thread-3 is incrementing...
Thread Thread-1 is incrementing...
Thread Thread-3 is incrementing...
```

4. Building Mutexes in a Multiprocessor Setting (Practical Section)

Goal:

Demonstrate how mutexes help coordinate access to shared resources in a multiprocessor environment.

Instructions:

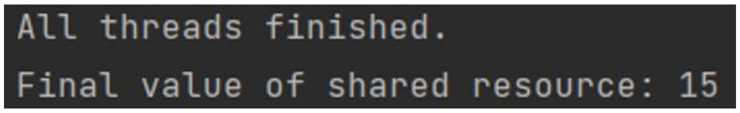
Write a new Python file called `mutex_example.py`.

Create a class representing a shared resource, including a method that will be protected by a mutex.

Spin up several threads that try to access this resource at the same time.

Use Python's `threading` module to implement mutexes to avoid race conditions.

Run the script and observe how mutexes affect the system's behavior.



```
All threads finished.
Final value of shared resource: 15
```

Conclusion:

This lab offered hands-on practice with multiprocessor systems and the critical role of synchronization. By simulating cache coherence and implementing mutex-based locking, you've gained valuable insights into the challenges and solutions involved in managing shared-memory multiprocessors under modern operating systems.