UNIVERSITY OF ENERGY AND NATURAL RESOURCES

SCHOOL OF SCIENCES

DEPARTMENT OF INFORMATION TECHNOLOGY AND DECISION SIENCES

COURSE: OPERATING SYSTEMS

NAME: ISSAH ABDUL-SALIM BORESA

I**NDEX NUMBER**: UEB3211422

CLASS: Level IT A

Deadlock in Operating Systems - Classwork Solutions

1. Explain the concept of deadlock by drawing parallels to real-life scenarios, providing three concrete examples to illustrate its occurrence and impact.

A deadlock is a situation in an operating system where two or more processes are unable to proceed because each is waiting for a resource held by another. This results in a state of indefinite waiting, preventing any progress. Deadlocks typically occur in multitasking environments where multiple processes compete for system resources such as CPU time, memory, and input/output devices.

Real-life Examples of Deadlock:

1 Traffic Jam at a Busy Intersection
- The image shows a chaotic traffic jam where vehicles and motorcycles are blocking each other in multiple directions, creating **gridlock**. This happens because **no vehicle can move forward without another moving first**, leading to a **deadlock situation** where all are stuck

Reference: Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems (4th ed.). Pearson.

2. Can deadlock occur on a uniprocessor system? Support your position with cogent reasons.

Yes, deadlock can occur on a uniprocessor system even though only one process executes at a time. This happens when multiple processes require resources in a way that causes circular waiting.

Reasons Why Deadlock Can Occur on a Uniprocessor System:

1. **Resource Locking:**
   - If a process holds a resource, such as a file, while waiting for another resource, such as memory, and another process holds that memory while waiting for the file, both processes get stuck indefinitely.

2. **Synchronization Issues:**
   - In multithreading environments, if two threads attempt to acquire locks in different orders (e.g., one locks Resource A then B, while the other locks B then A), they may end up waiting indefinitely.

3. **Circular Wait Condition:**
   - If a set of processes form a closed chain where each process waits for a resource held by the next, deadlock occurs, regardless of whether the system is uniprocessor or multiprocessor.

Reference: Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.

3. How can different deadlock prevention and resolution strategies be optimized to ensure efficient resource allocation across various computing environments, and what trade-offs exist between system performance, resource utilization, and process fairness?

Deadlock prevention and resolution strategies ensure efficient resource allocation in computing environments. However, each strategy involves trade-offs between performance, utilization, and fairness.

Strategies for Optimizing Deadlock Prevention and Resolution:

1. **Deadlock Prevention**:
   - Removing one of the four necessary conditions (mutual exclusion, hold-and-wait, no preemption, circular wait) to ensure deadlocks do not occur.
   - Example: Avoiding circular wait by enforcing a strict order in which processes request resources.

2. **Deadlock Avoidance**:

  - Algorithms such as the Banker's Algorithm check if granting a resource request leads to a safe state.
  - Example: The system maintains records of resource requests and only grants them if a safe sequence is ensured.

3. **Deadlock Detection & Recovery:**
  - The system periodically checks for deadlocks and resolves them through rollback or termination.
  - Example: Killing one of the processes involved in a deadlock to free resources.

 Trade-offs:
- **System Performance vs. Resource Utilization:** Preallocating resources reduces the risk of deadlocks but can lead to inefficient use of resources.
- **Fairness vs. Performance:** Terminating a process to resolve deadlock can be unfair but necessary for system stability.
- **Efficiency vs. Complexity:** Advanced deadlock avoidance techniques improve allocation but increase computational overhead.

Reference: Stallings, W. (2017). Operating Systems: Internals and Design Principles (9th ed.). Pearson.

4. Consider the scenario: Process A holds a memory page and requests another page. Process B holds the required page and needs the page held by Process A. Neither process can proceed. Analyze the scenario and indicate what kind of resource(s) that has been blocked and how can this be resolved.

Analysis of the Scenario:
- The resources involved are **memory pages**, which are non-preemptable resources.
- Both processes are in a circular wait situation where Process A holds a page that Process B needs, and vice versa.

Resolution Strategies:
1. Preemption:
  - The operating system could forcibly take a memory page from one process and allocate it to another.
  - Trade-off: May cause inconsistency or require process rollback.

2. Process Termination:
  - One of the processes is forcibly terminated and restarted.
  - Trade-off: This can result in loss of unsaved data.

3. Resource Ordering:

   - Enforcing a strict order in which memory pages are requested ensures that circular waits do not occur.
   - Trade-off: May require modification of system design.

Reference: Gagne, G., Galvin, P. B., & Silberschatz, A. (2018). Operating System Concepts (10th ed.). Wiley.