**💻 Operating Systems (OS) – Detailed Notes**

**1. Process vs Thread**

**🔹 Process:**

A process is an executing instance of a program. Each process has its **own memory space, file handles, code, and data**. Processes are **isolated** from each other and do not share memory directly.

**🔹 Thread:**

A thread is a **lightweight subunit of a process**. Multiple threads can exist within the same process and **share the same memory** and resources, such as open files and variables.

**✅ Key Differences Between Process and Thread:**

|  |  |  |
| --- | --- | --- |
| Feature | Process | Thread |
| Memory | Own address space | Shares address space with peer threads |
| Communication | Inter-process communication (IPC) needed | Can communicate directly |
| Switching | Costly (more overhead) | Faster (less overhead) |
| Isolation | Processes are isolated | Threads are not isolated |
| Failure impact | One process crash doesn’t affect others | Thread crash may affect process |

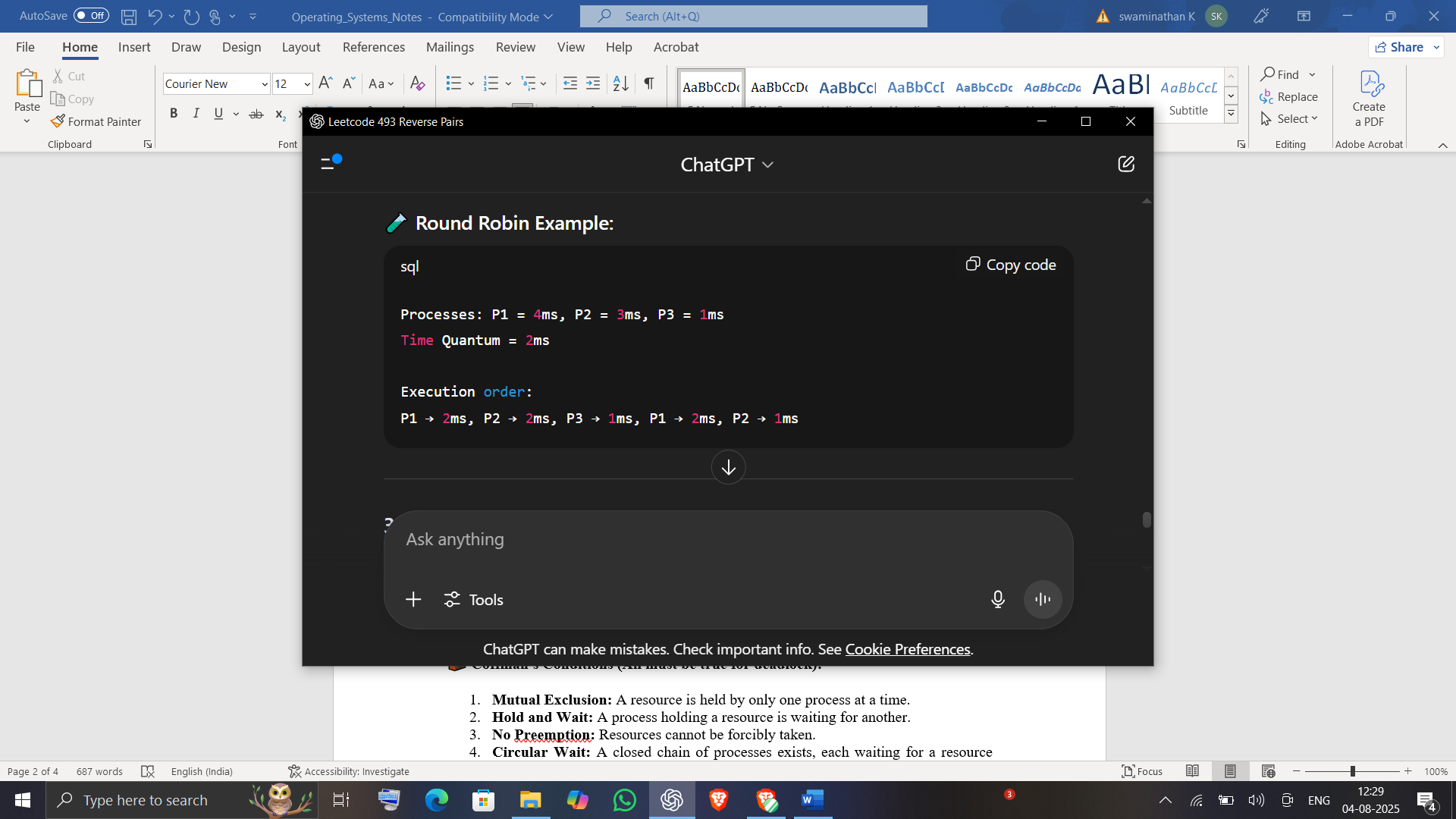
**2. CPU Scheduling Algorithms**

CPU Scheduling is the method by which the operating system **decides which process gets to use the CPU** and for how long. This is essential for **multiprogramming**.

**🔹 Key Algorithms:**

* **FCFS (First-Come, First-Serve):** Non-preemptive. Executes processes in the order they arrive. Simple but causes **convoy effect**.
* **SJF (Shortest Job First):** Non-preemptive. Selects the process with the **least burst time**. Can cause starvation.
* **Round Robin (RR):** Preemptive. Each process gets a **fixed time slice (quantum)** in circular order.
* **Priority Scheduling:** Assigns priority to each process. Can be preemptive or non-preemptive. May lead to **starvation** (solved by aging).

**🧪 Round Robin Example:**



**3. Deadlocks**

A **deadlock** occurs when two or more processes are **waiting indefinitely** for resources held by each other. No progress is possible.

**🧱 Coffman's Conditions (All must be true for deadlock):**

1. **Mutual Exclusion:** A resource is held by only one process at a time.
2. **Hold and Wait:** A process holding a resource is waiting for another.
3. **No Preemption:** Resources cannot be forcibly taken.
4. **Circular Wait:** A closed chain of processes exists, each waiting for a resource held by the next.

**✅ Deadlock Prevention Strategies:**

* **Avoid Hold and Wait:** Request all needed resources at once.
* **Preemption:** Take resources from waiting processes.
* **Avoid Circular Wait:** Impose a resource ordering.
* **Deadlock Avoidance (e.g., Banker's Algorithm):** Allocate resources only if they lead to a safe state.

**4. Paging vs Segmentation**

Memory management ensures efficient use of RAM by allocating and deallocating memory to processes.

**🔹 Paging:**

* Memory is divided into **fixed-size blocks**: Pages (logical) and Frames (physical).
* Pages are mapped to frames using a **page table**.
* Eliminates **external fragmentation** but may cause **internal fragmentation**.

**🔹 Segmentation:**

* Divides memory into **logical segments** like code, data, stack.
* Segments are of **variable length**.
* Reflects **logical organization of the program**.

|  |  |  |
| --- | --- | --- |
| Feature | Paging | Segmentation |
| Size | Fixed | Variable |
| View | Physical | Logical |
| Fragmentation | Internal | External |
| Table Used | Page Table | Segment Table |

**5. Synchronization (Semaphore vs Mutex)**

When multiple processes/threads access shared data, **race conditions** can occur. Synchronization techniques ensure **mutual exclusion and correct execution**.

**🔹 Semaphore:**

* Integer value used for signaling.
* Two types: **Binary Semaphore** (0 or 1) and **Counting Semaphore**.
* Common operations: wait() and signal().
* Can be used to manage **access control** for multiple processes.

**🔹 Mutex (Mutual Exclusion):**

* A lock mechanism used to protect critical sections.
* Only the thread that locked it can unlock.
* Used when **only one thread** must access a resource.

|  |  |  |
| --- | --- | --- |
| Feature | Semaphore | Mutex |
| Resource type | Can manage multiple | Only one at a time |
| Ownership | No concept | Has ownership |
| Usage | Signaling mechanism | Lock mechanism |

**📊 Common Operating System Diagrams (Visual Suggestions):**

* Process State Transition Diagram
* RR CPU Scheduling Queue
* Deadlock Resource Allocation Graph
* Paging Address Translation

**💬 Common Interview Questions**

1. What is the difference between a process and a thread?
2. Explain the working of Round Robin scheduling.
3. What are the four Coffman's conditions for deadlock?
4. What is the difference between paging and segmentation?
5. How does a semaphore differ from a mutex?
6. What are starvation and aging in CPU scheduling?
7. How does the OS handle context switching?
8. Describe a real-life analogy for deadlock.s