# Chapter 5 - Concurrency



1. **Advantages to Special Machine Instructions**
   Works on uniprocessor or shared-memory multiprocessor

   Simple and easy to verify

   Permits multiple critical sections, each with its own variable
2. **Binary semaphore**
   only two values, 0 and 1, but equally powerful as the general semaphore. Like a flag.
3. **Competition poses three problems**
   - Mutual Exclusion
   - Deadlock
   - Starvation
4. **Concurrency**
   -concurrent activities
   - happening at the same time
5. **Concurrency arises in each of these kinds of systems:**
   -Multiprogramming
   -Multiprocessing
   - Distributed Processing
6. **Concurrency occurs in the following contexts:**
   -Multiple applications
   -Structured Applications
   -OS Structure
7. **Counting semaphore**
   a general semaphore with n values.
8. **Deadlock**
   example: two processes want two resources but each owns only one.
9. **Distributed processing**
   multiple processes on multiple distributed systems.
10. **Drawbacks to special machine instructions**
    Busy-waiting consumes processor time

    Starvation possible since the process that runs next is arbitrary.

    Deadlock is possible if p1 enters the critical section and is interrupted by a higher priority process p2. p2 cannot enter due to p1, and p1 will not be scheduled due to p2.
11. **Finite Buffer Solution**
    can be constructed by adding a semaphore to keep track of the buffer size.

    A semaphore "e" is used to keep from inserting into a full buffer.
12. **Hardware**
    Disable interrupts or have special machine instructions to support mutual exclusion
13. **Interrupt Disabling**
    In a uniprocessor system, processes cannot have overlapped execution, but only interleaved execution.

    Interleaving occurs when a process is interrupted.

    Disabling interrupts lets a process complete its critical section without worrying about being interrupted and switched to another process.
14. **A monitor is a software module with these chief characteristics:**
    Local data variables are accessible only by the monitor

    Process enters monitor by invoking one of its procedures

    Only one process may be executing in the monitor at a time, others are suspended waiting for the monitor to become available
15. **Multiple Applications**
    Running at once
16. **Multiprocessing**
    multiple processes on a multiprocessor system.
17. **Multiprogramming**
    multiple processes on a uniprocessor system (also called multitasking).
18. **Mutex**
    also known as a mutual exclusion lock, it's a concept related to a binary semaphore.

    a program object that is created so that multiple program thread can take turns sharing the same resource, such as access to a file.

    Usually, a ____ requires the process that locks it to be the one that unlocks it.

    This differs from a binary semaphore, where one can wait for another to signal it.
19. **Mutual Exclusion**
    - Critical resource: only one process at a time can use it. Example is a printer.

    - Critical sections: portion of program using a critical resource. Only one program at a time is allowed in its critical section.
20. **OS/Language**
    Let the OS or programming language offer support for this capability

| | | |
|---|---|---|
| 21. | **OS Structure** | OS made of multiple pieces running at once |
| 22. | **Problems with Interrupt Disabling** | Degrades multitasking performance.<br><br>Doesn't work on multiprocessor system where multiple processes execute at once regardless of interrupts. |
| 23. | **Race Condition** | occurs when multiple processes or threads read and write data items so that the final results depends on the order of execution of instructions. |
| 24. | **A semaphore...** | is a variable with an integer value<br><br>may be initialized to a nonnegative number<br><br>wait(s) operation decrements the value<br><br>signal(s) operation increments value |
| 25. | **Semaphores** | an OS or language mechanism to support concurrency. (Proposed by Dijkstra in 1965) |
| 26. | **Software** | let the applications enforce it themselves through coding |
| 27. | **Starvation** | A process competing for a resource never receives it due to scheduling of other processes. |
| 28. | **Strong semaphore** | queue uses FIFO order. No starvation. |
| 29. | **Structured Applications** | Applications made of multiple pieces running at once |
| 30. | **Three cases of Process Interaction** | Processes unaware of each other<br>-Independent processes (competition)<br><br>Processes indirectly aware of each other<br>-Shared resource (cooperation)<br><br>Process directly aware of each other<br>-Communicate with each other (cooperation) |
| 31. | **Three General Approaches to achieving Mutual Exclusion** | -Software<br>-Hardware<br>-OS/Language |
| 32. | **Two ways hardware can support mutual exclusion:** | 1) Interrupt disabling<br>2) Special machine instructions |
| 33. | **Weak semaphore** | no queue ordering. Could have starvation. |
| 34. | **What problems does concurrency pose to the OS?** | Keep track of various active processes (use PCBs and other structures).<br><br>Allocate resources (CPU, Memory, Files, I/O devices) to each process.<br><br>Protect data and resources of each process from interference from other processes.<br><br>Process results must be independent of the speed at which execution occurs relative to other processes (the topic of this chapter). |