Module 3

Concurrency in OS Design

▼ What is concurrency?

when multiple computations are happening at the same time, e.g. several process threads are running and require the same resources

▼ How do threads behave in single processor systems?

threads from different processes take turns being executed but to the user appear to be running at the same time

▼ Concurrency vs. Parallelism

- Parallelism: simultaneous execution of multiple processes
- Concurrency: create a good structure of segmenting a process into independent pieces and coordinating among them

▼ What does concurrent processing look like in a multiprogramming and single processor system?

when threads from different processes take turns, the processes are interleaved or layed

▼ What can interleaving cause?

interleaving (with or without an overlap in the layers) can remove the ability to predict the execution time for each process (since threads from each process must wait for their turn to run)

▼ Example of parallelism

- Suppose that you need to print out two documents.
- One document is 5 pages long, and the other one is 12 pages long.
- If you had two printers, you could print one document on one printer, and the other one on the other printer **at the same time**.

▼ Example of concurrency

- Suppose that you need to print out two documents.
- One document is 5 pages long, and the other one is 12 pages long.
- Imagine you have one printer and you want to print them concurrently.
- **Do you print one page of each document, then the next page?** How about one page of the shorter document, then two pages of the longer one? When one page gets jammed, both documents are stuck, waiting until the problem is fixed. What happens when it is fixed? Does the problematic page get printed again, or is it the other document's turn? These are just some of the things that need to be considered with concurrent processing in operating systems.

▼ What are critical sections?

parts of the program where the shared resource is accessed must be protected and are not to be executed by more than one process at a time

▼ What conditions must a critical section solution possess?

- Mutual exclusion: allows only one process to have access to a resource or perform a function, at a time
- Progress: each process will get its turn in the critical section at some point
- Bounded waiting: a bound exists on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

▼ How can mutual exclusion be achieved through the operating system?

- semaphores
  - an interger value for signaling among processes with three possible atomic operations: initialize, decrement and increment
- monitors

    - a higher-level construct that the semaphore
    - an object or set of programs/procedures that is easier to work with and validate
    - ME is secured by allowing only one process to execute the monitor’s code at any given time
- message passing
    - similar to semaphore
    - a way for processes which do not share data to exchange information, by sending and receiving messages that include a blocking policy which instructs whether the process will wait or continue executing when sending or receiving a message

▼ What is deadlock?

- can happen when implementing mutual exclusion
- processes are blocked because each process is holding a resource and waiting for another resource that’s acquired by some other processes

▼ How to handle deadlock?

- Ignore it. If it occurs, reboot the system (Windows and UNIX do this)
- Detect it. Check for deadlocks and terminate the processes involved or reallocate resources to other processes so that the deadlocked processes cannot use them
- Prevent or avoid it. Create conditions that do not allow the system to enter into a deadlocked state.