1 Concurrency

Threads and processes **execute concurrently** or in **parallel** and can share resources. *Multi-programming* **improves** system utilisation.
A process/threads can be **interupted** at any point in time (timer, I/O). The process state (including registers) is **saved** in the *process control block*
The outcome of programs may become **unpredictable**

- Sharing data can lead to **inconsistencies** (e.g when interupted whilst manipulating data)
- The outcome of execution may depend on the **order** in which instructions are carried out.

2 Race conditions

A *race condition* occurs when multiple threads/processes access **shared** data and the result is dependent on the **order** in which the instructions are interleaved.

3 Concurrency within the OS

3.1 Data structures

Kernels are preemptive these days.

- **Multiple** processes/thrads are running in the kernel
- Kernel processes can be **interupted** at any point

The kernel maintains **data structures**, e.g. process tables, memory structures, open file lists, etc.

- These data structures are accessed **concurrently/in parallel**
- These can be subject to **concurrency** issues

The OS must make sure that interactions within the OS **do not** result in *race conditions*

3.2 Resources

Processes **share** resources, including memory, files, processor time, printers, I/O devices, etc. The OS must

- The operating system must provide a *locking mechanisms* to implement support *mutual exclusion* and prevent *starvation* and *deadlocks*
- Allocate and deallocate these resources safely (i.e. avoid interference deadlocks and starvation)

3.3 Critical Sections, Mutual Exclusion

Any solution to the *critical section* problem must satisfy the following **requirements**:

- *Mutual exclusion* only one process can be in its critical section at any one point in time
- **Progress** any process must be able to enter its critical section at some point in time. Processes/Threads in the **remaining code** do not influence access to critical sections.
- **Fairness/bounded waiting** processes cannot be made to wait indefinitely

These requirements have to be satisfied, **independent of the order** in which sequences are executed. Approaches for mutual exclusion can be:

- Software based: *Peterson's solution*
- Hardware based `test_and_set ()`, `swap_and_compare()`
- Based on: Mutexes, Semaphores, Monitors

In addition to mutex, *deadlocks* have to be prevented.

3.4 Deadlocks

Four conditions must hold for deadlocks to occur:

- Mutual exclusion
- Hold and wait condition: a resource can be held whilst requesting new resources
- No preemption: resources cannot be forcefully taken away from a process
- Circular wait there is a circular chain of two or more processes waiting for a resource held by the other processes.

Reference section

Multi-programming

a rudimentary form of parallel processing in which several programs are run at the same time on a uniprocessor. Since there is only one processor, there can be no true simultaneous execution of different programs. Instead, the operating system executes part of one program, then part of another, and so on.

Process control block(PCB)

The role of the PCBs is central in process management: they are accessed and/or modified by most OS utilities, including those involved with scheduling, memory and I/O resource access and performance monitoring.

Critical section a set of instructions in which **shared** resources between processes/threads (e.g. variables) are **changed**

Mutual exclution(Mutex)

a program object that prevents simultaneous access to a shared resource. This concept is used in concurrent programming with a critical section. Processes have to get **permission** before entering their critical section. (request a lock, hold the lock, release the lock)

Peterson's solution

a concurrent programming algorithm for *mutual exclusion* that allows two or more processes to share a single-use resource without conflict, using only shared memory for communication.

Deadlock

a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.