Week 4

Local Object Destruction Order

In C++, the destruction order of local objects is the reverse of their construction order. This behavior is dictated by C++'s scoping rules, where the last declared object is destroyed first. This is crucial for managing resources like memory, file handles, or locks, ensuring that all resources are safely released in reverse order upon exiting a function or scope.

Fire and Forget

"Fire and Forget" is a common programming approach, especially useful for tasks that are initiated and then not monitored or awaited by the caller. This method helps reduce complexity in managing thread states and is suitable for operations where immediate feedback or outcomes are not necessary, such as logging or sending notifications.

Race Condition

A race condition occurs in multi-threaded programs when two or more threads access shared data without proper synchronization and attempt to modify it simultaneously. This leads to unpredictable outcomes and program errors. Understanding and preventing race conditions is a core challenge in concurrent programming.

Lifecycle of Local Variables with Threads

In multi-threaded programs, special attention must be given to the lifecycle of local variables accessed by multiple threads. Particularly when creating threads, it is essential to ensure that local variables passed to threads are correctly referenced using `std::ref` or `std::cref` to prevent issues caused by the destruction of local variables while still in use by child threads.

Context Switching and Thread Performance

Frequent context switching can significantly reduce thread performance as each switch requires CPU time to save and load thread states. In designing multi-threaded applications, minimizing context switching by reducing the number of threads or using more efficient thread synchronization mechanisms is advisable.

Thread Starvation and Bounded Waiting

Thread starvation describes a situation where a thread is unable to access necessary resources to execute its tasks, possibly due to the activity of other threads or unfair system scheduling policies. Bounded waiting is a fairness technique that ensures each thread can access the resources it needs within a limited time, thereby preventing starvation.

Lock-Free Programming

Lock-free programming is a technique that avoids using mutexes for thread synchronization, instead ensuring safe access to shared data among multiple threads through atomic operations. Lock-free programming can enhance parallel performance of a program but often involves a more complex programming model and is more challenging to implement correctly.

Software Transactional Memory (STM)

Software Transactional Memory (STM) is an abstraction for managing access to shared memory, allowing multiple threads to execute transactions on memory that can automatically commit or rollback in case of conflicts. STM aims to simplify concurrent programming, offering a higher-level synchronization mechanism that is easier to use than traditional locks.

Execution of Asynchronous Tasks

Asynchronous tasks do not always mean parallel execution; they may run sequentially on the same CPU core. Asynchronicity refers to the non-blocking nature of task initiation and execution, allowing other tasks to be performed or I/O operations to be processed simultaneously.

Difference Between async and thread

Both std::async and std::thread can be used to create new threads in C++. Using std::async makes it easier to obtain the return value of a thread and manage its execution policy (e.g., deferred launch). In contrast, std::thread provides lower-level control, suitable for scenarios that require detailed management of thread lifecycles.

Fold Expression

Introduced in C++17, Fold Expression allows for the expansion of template parameter packs and performing operations such as addition, logical AND, or

logical OR on the elements. Fold expressions simplify the handling of variadic templates.