Assignment (Video- 4 to 6): Database System Impl. (COP6726)

Name: Vikas Chaubey**, UFID:** 3511 5826**, Email:** vikas.chaubey@ufl.edu

1) **Java Object Referencing:** In java object referencing is done through using technique of "Aliasing". That means in java when a new object is created, and it is kept in the memory. All the variables which have access to that object carry the memory reference of that object. All the created objects are generally kept in a heap even all the class objects which define classes are also kept in heap memory. Aliasing is the technique where multiple variables point to the same object because they carry the same reference to the object in the memory.

2) **Java Reference Counts, Object Deletion and Garbage Collection:** Since Java uses aliasing to keep object references among variables it can create various issues in the program if the object deletion is not handled properly. For example, when multiple variables point at the same object i.e. this object is being used by different sources in that case if one source deletes the object then it might create several inconsistencies. hence in Java object deletion is handled using a special mechanism. where the java environment keeps track of counts of references of objects currently have. Once object references reach the count of 0, Java garbage collector assumes that the object is no longer in use and deletes the objects.

3) **Java Performance with Aliasing and garbage collector:** Java aliasing and garbage collection give good performance as C++ in case of single thread but in case of multiple threads, reference counts need to be thread safe, hence in this scenario Java performance is very low compared to C++.This happens because in order to safe guard resources , synchronization of the resource usage is done in Java which adds additional overhead in performance.

4) **Pointers in C++ and Object Deletion:** Unlike Java there is no concept of garbage collector in C++. Also, unlike java when an object is created in C++, they are generally saved in the stack. The referencing of the objects in C++ is done through pointers. Hence all the variables referring to the same objects have pointer to access the objects, if the object is deleted by any source then the program can have serious inconsistencies.

5) **Swapping Paradigm in C++:** Aliasing of objects creates the deletion problem with objects , on the other hand there is performance overheads associated with synchronization hence in C++ in order to address both of these issues swapping paradigm could be used which can take care of both of these problems without creating programming inconsistencies and performance issues. Under this paradigm the object pointers are swapped among the variables that means at any given time only one variable has the object pointer and other variables don't, hence the objects would be accessed by one resource at a time. This resolves the deletion inconsistencies while still maintaining the performance

6) **Information Exchange Among Threads Using Message queues:** The information could be exchanged among threads by using global resources such as global variables or objects which are available to all threads and which could be used by them. But it is not a good approach to program multithreaded application because it does not provide good abstraction and can generate various inconsistencies. Hence Message queues could be used for this purpose. Thread could write messages to queues and other threads could read those messages from the queue in a producer consumer fashion, this way inconsistencies could be avoided.

7) **Writing Efficient Code and Processor Architecture:** As the advancement of computer technology the compilers and processors have become more intelligent that means they incorporate their own mechanisms and intelligence in order to execute the code written by developers faster and efficiently. Hence in order to write efficient code developers must understand what kind of mechanism these compilers and processors incorporate so that we don't make mistakes while writing the code which can make the program execution slower. That is why it is important to understand the processor architecture. One can write efficient code which executes faster by simply following the benchmarks against which the processors are tested.

8) **Pipelining in Processors and Pipeline Flushes:** Processors executes the instructions one by one as it flows these instructions flows into execution unit from the registers. These instructions are executed in one clock cycle but in order to execute instruction in this fast manner, processor incorporates a mechanism of pipelining , where each operation is divided into several steps and then a execution pipeline is formed , when one step is completed instruction is moved to next step while a new instruction enters for execution in previous stage. That way the execution is optimized in processors. In order to write fast executing programs, one needs to take under consideration the unpredictable "If" statements included in program which can cause pipeline flushes and these flushes could significantly reduce program performance.

9) **Performance Counters:** Performance counters are the event counters in a processor, these counters could be programmed to count certain events. By counting these events the performance of the program execution could be calculated. For example, these counters could be tasked to count if clauses where they were actually predictable by the processor, which can give an idea about pipeline flushes. This way we can evaluate program performance.

10) **Memory Hierarchy:** Processors implement various levels of memory in order to efficiently execute the program instruction. If a processor has to fetch instruction or data from the closest memory level, then it takes less time to execute the program and if the instruction or data is fetched from farthest point then it takes more time to execute program code. The various memory levels are as follow:

    1) **Registers**: This is the closest memory level for the processor. Registers provide fastest possible access. It takes one CPU cycle. Registers could be up to few thousand bytes in size.
    2) **Cache**: Cache is the second closest memory level to the processor after register. A CPU cache is a hardware cache used by the central processing unit (CPU) of a computer to reduce the average cost (time or energy) to access data from the main memory. There are various levels of cache memories available such as: L0 cache, L1 cache, L2 cache, L3 and L4 caches. As increase the cache levels and move away from processor the cache capacity is increased but the access speed is decreased.
    3) **Main or Primary Memory**: This is the primary memory of the CPU where all the data which is being processed is kept. Data fetched from hard disk or magnetic disks is kept in primary memory for processing. Generally, the size of primary memory could be in Gigabytes and they have high access times compared to registers and cache memories.