Abhisth Chatterji

500084948

R2142201733

B-3 6th Semester

BTech CSE (CCVT) Non-Hons

Week 3: Thread Programming

Submitted to: Saurabh Shanu Sir

Use of threads in my project

- Create a main thread to control the application's general flow, including the creation of subthreads.
- Create a thread to manage the application's IAM, including user authentication and permission. While the rest of the application is still running, this thread can run in the background.
- Create a thread to handle data encryption and decryption for the KMS component of the application. Once the user has been successfully authorised by the IAM thread, this thread may be started.
- Create a thread to handle all S3-related tasks for the application, such as downloading and uploading files to S3 buckets. Once the data has been securely encrypted or decrypted by the KMS thread, this thread can be launched.
- The DynamoDB portion of the application, including the generation and manipulation of database tables, should be handled by a separate thread. Once the S3 thread has successfully uploaded data to an S3 bucket, this thread may be started.
- Create a thread to manage the Athena portion of the application, which includes accessing data from DynamoDB tables and S3 buckets. Once the DynamoDB thread has successfully constructed the tables and loaded the data into them, this thread can begin.
- Create a thread to handle the CloudTrail section of the programme, which includes recording AWS API calls. Once the Athena thread has successfully processed queries and produced logs, this thread can be launched.
- Create a thread that manages the application's CloudWatch functionality, including logging and monitoring. Once the CloudTrail thread has successfully generated logs, this thread can be started.

Thread Programming

The creation of software applications that utilise threads to carry out numerous activities concurrently is referred to as thread programming. A thread is a small, autonomous execution unit that can run in a programme and share resources with other threads.

By allowing various components of the application to run simultaneously rather than waiting for one operation to finish before starting another, thread programming can enhance the performance of the application. For instance, a web server application can handle several requests concurrently by using threads, which speeds up response times and lowers latency.

The following are some essential ideas and methods for thread programming:

Synchronization: Because threads share resources like memory and data, it's critical to control how these resources are accessed in order to avoid race situations and other concurrent problems. To synchronise access to shared resources, synchronisation mechanisms like locks, semaphores, and monitors can be employed.

Java, Python, and C++ are a few examples of programming languages that allow for the development and control of threads. APIs for creating and managing threads are provided by libraries like the POSIX Threads library and the Java Concurrency API.

A thread pool is a collection of threads that can be utilised to carry out activities concurrently. Libraries like the Java Executor framework and the.NET ThreadPool class can be used to create and manage thread pools.

Livelocks and deadlocks: A deadlock occurs when two or more threads get stalled while one is awaiting the release of a resource that the other thread is holding. When two or more threads are unable to advance because their states are constantly shifting in reaction to one another's actions, this is known as a livelock.

Performance tuning: Balancing the number of threads with the available resources, eliminating needless synchronisation, and reducing context switching are all steps in improving thread programming efficiency.

Relationships between thread and process

A software instance that is being run by a computer is known as a process. Its own memory area, system resources, and execution environment make it a self-contained unit of execution. The memory space of other processes cannot be accessed without utilizing specialized inter-process communication mechanisms since each process operates in its own memory space.

A thread is a component of a process that can work along with other threads running in the same process at the same time. The memory and system resources used by the process that created a thread are shared by all threads. The heap memory is shared by all threads, but each has its own stack and register set.

Processes and threads are related in that a process may include one or more threads. A single thread of execution and the process itself serves as the execution context in a single-threaded process. Several threads of execution share the same process resources and memory space in a multi-threaded process. It is simpler to share data and coordinate execution when threads within a process are able to communicate with one another directly rather than through inter-process communication protocols.

Parallel computing approaches using threads

When employing threads for parallel computing, work is divided into smaller subtasks and carried out concurrently on several threads within a single process. When employing threads for parallel computing, there are two major strategies:

Multiprocessing with shared memory allows many threads to access data structures directly while sharing the same memory space. This method is frequently applied in programmes that need a lot of synchronization, including video rendering, database management systems, and scientific simulations.

Distributed memory multiprocessing: In this method, each thread has its own memory space and interacts with other threads using message passing or remote procedure calls, two inter-process communication techniques. This method is frequently employed in large-scale data processing applications like distributed databases, data analytics, and web server processing.

To prevent race situations and deadlocks, both strategies necessitate careful management of shared resources like locks and semaphores. Using several cores or processors, parallel computing with threads can increase performance and be used for a range of tasks, including scientific simulations, data analysis, and web server processing.

Multithreading using cloud application platforms

Platforms for cloud applications can leverage multithreading to increase scalability and performance. Scalable and flexible computing resources are offered through cloud application platforms like Amazon Web Services (AWS) and Microsoft Azure, which are simple to provision and operate.

In order to make the best use of available resources and enhance overall programme performance, multithreading can be used in cloud applications. Multithreading, for instance, could be used by a web application operating on a cloud platform to manage numerous user requests concurrently, enhancing response times and decreasing latency.

Moreover, services and tools for managing multithreaded applications are offered by cloud application platforms. For instance, AWS Elastic Beanstalk supports Java applications that use the Java Concurrency API for multithreading. With the Task Parallel Library, Microsoft Azure supports multithreading in.NET applications.

Additionally, to enhance the use of multithreading in cloud applications, cloud platforms include functions like load balancing, auto-scaling, and containerization. For instance, containerization can be used to isolate and manage many threads within a single programme, while auto-scaling can be used to automatically add or remove computing resources based on demand.

Cloud Application Platform thread vs common threads

Threads are utilised in cloud application platforms to execute code concurrently, enhancing the speed and scalability of applications. Threads on cloud application platforms, however, differ from regular threads in a number of ways.

Resource management: Cloud application platforms offer a variety of resources, such as virtual machines or containers, that can be utilised to run threads. Based on the workload and demand of the application, these resources can be handled dynamically. Common threads, on the other hand, must share the same resources, such as CPU cycles and memory, on a single computer, which can cause contention and performance loss.

Scalability: The infrastructure offered by cloud application platforms is scalable and may be utilised to dynamically deploy and manage resources, including threads. As a result, programmes can scale vertically by adding more threads to a single system or horizontally by adding more threads across several machines. The resources of a given machine have an impact on the number of common threads it can support.

Redistributing workload and resources in the event of faults or interruptions is one way that load balancing and auto-scaling technologies offered by cloud application platforms can assist an application become more fault tolerant. On a single computer, common threads do not have the same level of fault tolerance and are more vulnerable to interruptions like hardware malfunctions or power outages.

Building Applications using cloud computing platform threads

Developing code that can run concurrently on several threads in a cloud environment is necessary when creating applications that use cloud application platform threads. To create applications using cloud application platform threads, follow these steps:

Choose a platform for cloud applications: Choose a cloud computing platform that supports multithreading and related features like load balancing and auto-scaling, such as Amazon Web Services (AWS) or Microsoft Azure.

Create your application with concurrency in mind: Determine which sections of your application may run concurrently and design those sections to use multiple threads. To manage shared resources and avoid race situations and deadlocks, use coding techniques like locks and semaphores.

Multithreading should be used in your code: To develop your concurrent code, use a multithreading-compatible programming language, such Java or C#. Use frameworks and libraries that allow multithreading, such as the.NET Task Parallel Library or the Java Concurrency API.

Use thread management services on cloud platforms: Use the load balancing and auto-scaling features offered by your cloud platform to control the resources required for running your multithreaded applications. With the use of cloud platform tools like AWS CloudWatch or Azure Monitor, you can track and examine the performance of your application.

Test and improve your application: Test your application under various loads and circumstances to find bottlenecks and potential areas for improvement. To simulate and test your application in real-world scenarios, use cloud platform technologies like AWS Elastic Load Testing or Azure Load Testing.

Video Presentation:
https://drive.google.com/file/d/1tosXZ0PVNj4QG3sxdf7gCmcgTsqBwTJ8/view?usp=sharing

GitHub Link: https://github.com/Ac-11/National_DnaDatabase

References

1. Rittinghouse, J. W., & Ransome, J. F. (2017, March 27). *Cloud Computing: Implementation, Management, and Security*. CRC Press. https://doi.org/10.1201/9781439806814

2. Herlihy, M., & Shavit, N. (2008, February 29). *The Art of Multiprocessor Programming*. https://doi.org/10.1604/9780123705914