A picture containing text

Description automatically generated

|  |  |
| --- | --- |
| Module code and name: | Concurrent Programming - 6BUIS011C |
| Assessment: | Coursework |
| Student ID: | 00012670 |
| Submission deadline: | 5.04. 2024 |
| Wordcount | 3000 |
| Results date and type of feedback |  |
| Result (Pass or Fail) |  |
| **Feedback** | |
|  | |

Table of Contents

[Introduction 2](#_Toc163253747)

[System design 3](#_Toc163253748)

[Synchronization primitives 3](#_Toc163253749)

[Concurrent techniques 5](#_Toc163253750)

[Diagrams 14](#_Toc163253751)

[Implementation 16](#_Toc163253752)

[Conclusion 33](#_Toc163253753)

[Reference 34](#_Toc163253754)

Introduction

In today's digital world, efficient file downloading is crucial for various applications, ranging from software updates to media content delivery. To simplify this process, it was present a comprehensive solution that includes a web service and a user-friendly client application. The project aims to simplify the file download through concurrent processing, progress tracking, and intuitive user interaction.

**Project Overview**

The project consists of two key components: a Web Service and a Windows Forms Client Application. The Web Service, implemented as a WCF service named DownloadService, serves as the backbone for managing download requests and coordinating concurrent downloads. The Windows Forms Client offers a user-friendly interface for initiating, monitoring, and controlling downloads.

System design

The project is structured into a client-server architecture:

1.The client side of the application is a Windows Forms application. It provides a user interface where users can input download URLs, target paths, and set download priorities. It also displays the progress of each download. The client communicates with the server using a WCF service client.

2. The server-side of the application is a WCF service that handles the actual downloading of files. It exposes several operations such as DownloadFile, GetDownloadProgress, PauseDownload, ResumeDownload, and CancelDownload. The server maintains a dictionary of download details for each active download, which includes information like progress, total file size, and status.

The application uses various synchronization primitives like Semaphore, lock and ManualResetEvent to manage concurrency and ensure thread safety.

Synchronization primitives

In concurrent programming, synchronization primitives are used to coordinate the execution of threads and ensure that they work together correctly. Different synchronization primitives are suited to different tasks and can have a significant impact on the behavior and performance of the program. Here's a brief overview of the synchronization primitives:

1. **Semaphore**

A semaphore is used to limit the number of threads that can access a certain resource or section of code at a time. In the project, it is used a semaphore to limit the number of concurrent downloads. This can help to prevent overloading the system with too many simultaneous downloads (BillWanger, 2022).

1. **ManualResetEvent**

A ManualResetEvent is used to block one or more threads until a signal is received (Stephen Toub, 2012). In the download manager project, it is used a ManualResetEvent to coordinate the start of downloads. This can help to ensure that downloads start in the correct order and that a new download doesn't start until the previous one has been properly set up.

1. **Lock**

The lock is used to ensure that one thread does not enter a critical section of code while another thread is in the middle of it (BillWanger, 2022). In the project, it is used a lock when updating the progress of a download. This can help to prevent race conditions where two threads try to update the progress at the same time, which could lead to incorrect progress values.

Each of these synchronization primitives has its own strengths and weaknesses:

Semaphores are useful for limiting concurrency, but they don't provide any guarantees about the order in which threads will enter the critical section. If the order of execution is important, a semaphore might not be the best choice (GeeksforGeeks, 2021).

ManualResetEvents are useful for signaling between threads, but they don't provide any protection against race conditions. If multiple threads are modifying a shared resource, need to use a different synchronization primitive to ensure safe access and prevent conflicts (Stack Overflow, n.d.).

Locks are useful for protecting critical sections of code, but they can lead to deadlocks if not used carefully. If a thread holding a lock tries to acquire the same lock again, or if two threads each hold a lock that the other wants, a deadlock can occur (GeeksforGeeks, 2017).

Concurrent techniques

Concurrent programming techniques can significantly enhance the performance and responsiveness of real-time applications (Pinho, n.d).

Here's how:

1. **Improved responsiveness**

In a single-threaded application, if a long-running task is being executed, the entire application can become unresponsive. With concurrent programming, long-running tasks can be moved to a separate thread, keeping the main thread (usually the UI thread) free and the application responsive (BairesDev Blog, 2023; NetNut, 2023).

1. **Better CPU utilization**

Modern computers have multiple cores. Concurrent programming allows applications to run multiple threads or tasks in parallel, using multiple cores and thus improving the overall performance of the application (BairesDev Blog, 2023; NetNut, 2023).

1. **Faster execution**

Tasks are independent and don’t rely on each other, executing them simultaneously can significantly reduce the overall time (BairesDev Blog, 2023).

1. **Real-time data processing**

Concurrent programming is crucial in real-time systems where it is needed to process incoming data while also performing other tasks. For example, in a real-time gaming server, one thread could handle incoming data from players, another could handle game logic, and another could handle sending data to players.

1. **Asynchronous I/O operations**

I/O operations (like reading from a file or a network request) can often be time-consuming. Concurrent programming allows these operations to be performed asynchronously, so the application can continue doing other work instead of waiting for the I/O operation to complete (BairesDev Blog, 2023).

However, concurrent programming also introduces complexity and potential issues like race conditions, deadlocks, and more. It's important to use synchronization primitives (like locks, semaphores, etc.) correctly to ensure that concurrent code is thread safe (Pinho, n.d).

Download manager project uses several concurrency techniques:

1. **Multithreading**

The project uses multiple threads to handle different downloads simultaneously. This is evident in the addToQueue\_Click method where a new thread is started for each download in the client-side.

A screen shot of a computer program

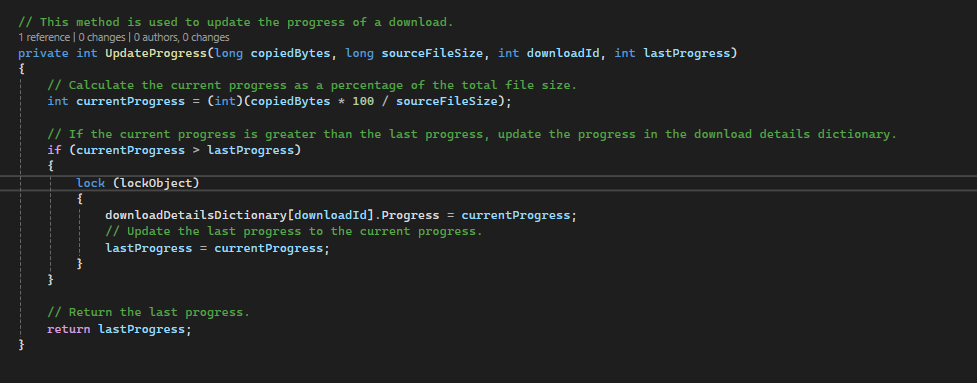
Description automatically generated

2. **Locks**

The lock keyword is used to ensure that only one thread can access the downloadDetailsDictionary at a time, preventing race conditions. This is used in methods like GetDownloadProgress, UpdateProgress, and DownloadFile on the server side.

A screenshot of a computer program

Description automatically generated



A computer screen with text on it

Description automatically generated

3. **ManualResetEvent**:

In the StartDownload method within Form1.cs, there's a ManualResetEvent called downloadResetEvent. Here's how it works:

A screen shot of a black screen

Description automatically generated

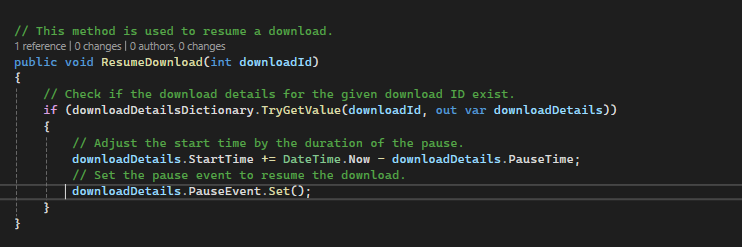
At the beginning of the download process, the code waits for certain conditions to be met before proceeding. This waiting is accomplished using downloadResetEvent.WaitOne(). This line of code essentially puts the current thread on hold until downloadResetEvent is signaled, indicating that it's okay to proceed with the download.

Once the necessary conditions are met and the download process is ready to start, downloadResetEvent is set to a signaled state using downloadResetEvent.Reset(). This allows the waiting threads to continue their execution, initiating the download process.

It's important to note that after each download starts, downloadResetEvent is reset to a non-signaled state. This means that any subsequent downloads will have to wait until the event is signaled again, ensuring that downloads occur in the correct sequence and avoiding any potential conflicts.

In addition, on the server side, manual reset event is used to pause and resume downloads. In the PauseDownload method, the Reset method is called on the PauseEvent to pause the download, and in the ResumeDownload method, the Set method is called to resume the download.

A screenshot of a computer program

Description automatically generated  


4. **CancellationToken**

The CancellationTokenSource and its associated CancellationToken are used in this project to provide a way to cancel ongoing downloads.

In .NET, a CancellationToken serves as a signal to distribute the notification that an operation should be cancelled. It’s commonly used in scenarios where tasks might be long-running, such as file downloads (Asryan, n.d).

Here's how it works in this project:

1. When a download starts, a CancellationTokenSource is associated with it. This association is stored in the Download Details object specific to that download.

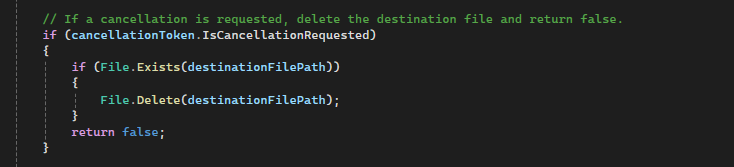
2. If the user decides to cancel the download, the CancelDownload method is called, which triggers the CancellationTokenSource.Cancel method.

3. The Cancel method sends a cancellation request to the associated CancellationToken. Any operation (like download process) that monitors this token can check the IsCancellationRequested property. If the property is true, the operation should stop and terminate it.

This mechanism provides a cooperative way for tasks to be cancelled. The task needs to periodically check if cancellation has been requested and then act on it. This is a clean and safe way to stop tasks, as it allows tasks to clean up resources and finish any necessary work before terminating (IEvangelist et al, 2022).

A computer screen shot of a program

Description automatically generated



5. **Semaphore**

The Semaphore class in C# serves as a mechanism to control how many threads can simultaneously access a shared resource or a pool of resources (Microsoft, 2024). In the context of a download manager project, it was initialized the semaphore with a count of 3. This means that up to three threads can enter the critical section (the part of the code that needs exclusive access) concurrently. Essentially, it ensures orderly and controlled access to the resource, preventing chaos and conflicts.

  
A screenshot of a computer program

Description automatically generated

When a download starts, the WaitOne method is called on the semaphore, which decrements the semaphore count. If the count is already at zero, the thread blocks until a spot becomes available (when Release is called).

The buttonClickSemaphore ensures that button clicks are processed sequentially, preventing any race conditions if the user clicks a button while a download is already in progress.

The downloadResetEvent is a ManualResetEvent that is used to block or unblock threads. When a download starts, WaitOne is called on the downloadResetEvent, which blocks the thread if the event is not set. After processing a button click, the event is reset using the ‘Reset’ operation. Following this, the ‘Release’ method is invoked on the buttonClickSemaphore, allowing the system to handle another button click.”.

In the CancelDownload method in the Form1.cs file, a semaphore is also used to control the number of simultaneous downloads. When a download is cancelled, the semaphore is released, allowing another download to start.

A screenshot of a computer program

Description automatically generated

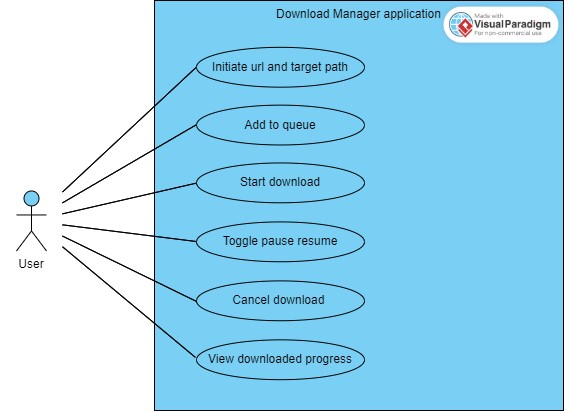
1. **Async/Await**: The async and await keywords are used to perform non-blocking operations, specifically when downloading a file in the DownloadFile method. This allows the UI to remain responsive while the download is in progress.

A screenshot of a computer program

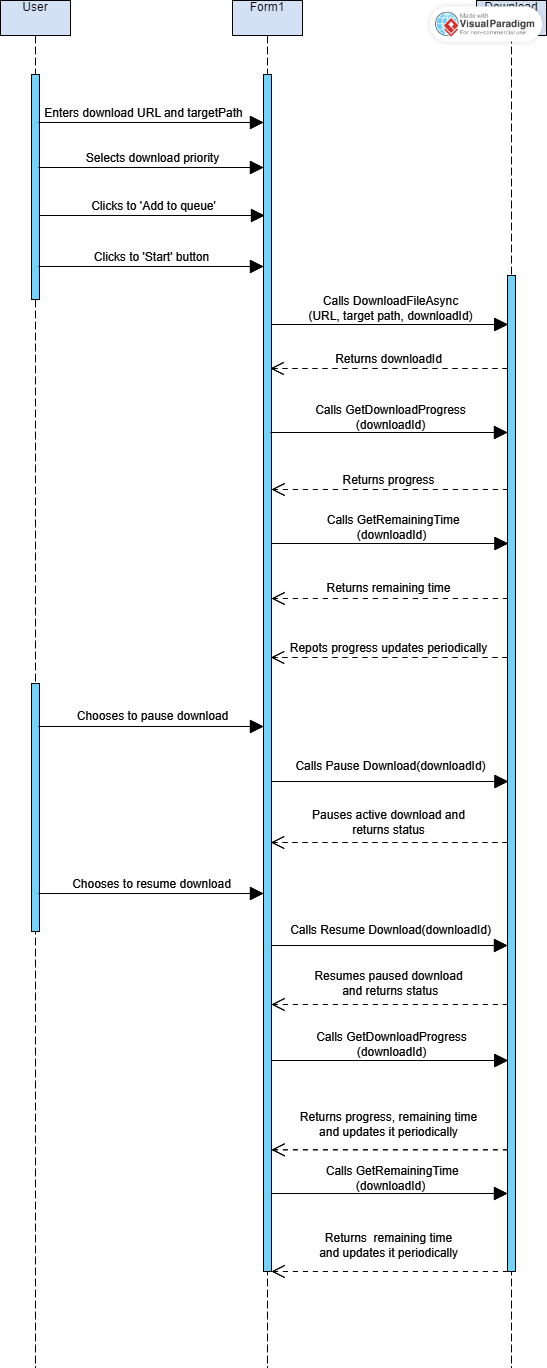
Description automatically generated

Diagrams

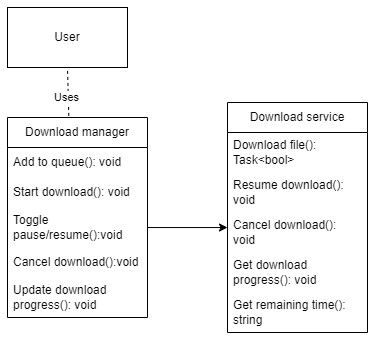
**Use case diagram**



**Sequence diagram**



**Class diagram**



**In this diagram:**

* The User is the actor who interacts with the system.
* The Download Manager is the main component of the system that the user interacts with. It provides methods to add downloads to the queue, start downloads, pause/resume downloads, cancel downloads, and update download progress.
* The Download Service is a service used by the Download Manager. It provides methods to download files, pause, resume, cancel, get the progress, and get the remaining time for a download.

Implementation

Software Development Life Cycle (SDLC) phases:

**Requirements gathering and analysis**

At this initial stage, the main task is to understand and document the project requirements. This implies a deep understanding of the functionality of the download service, the intended user interaction, as well as any relevant limitations or system requirements. The goal is to create a clear and detailed plan that will serve as a basis for subsequent development phases.

**Design**

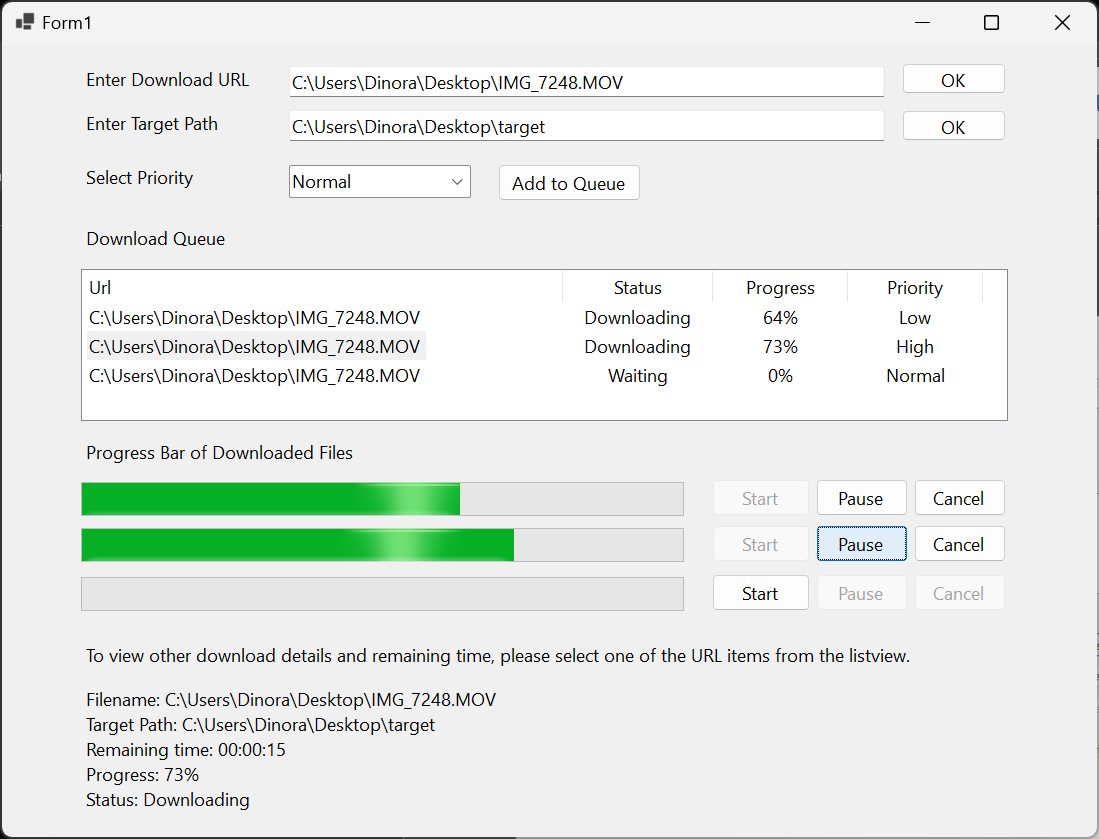
The Design phase of the Software Development Life Cycle (SDLC) involves defining the software's architecture, components, interfaces, and data for a system to meet specified requirements. It serves as a blueprint for the development and deployment process (Clark, 2022).

In the context of this project, the design phase would have involved the following:

1. **Defining the architecture**

The project is divided into two main parts - the client-side application (Form1.cs) and the server-side service (DownloadService.cs). The client-side application interacts with the user and sends requests to the server-side service, which handles the downloading of files.

1. **Designing the components**

The client-side application has several components such as buttons, text boxes, list view, labels and progress bars that allow the user to input the download URL, target path, and priority, start or pause the download, and view the download progress. The server-side service has a DownloadService class that provides methods for downloading files, pausing, resuming, and canceling downloads, and getting the download progress and remaining time.  
  
 

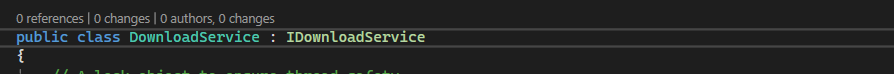
1. **Designing the interfaces**

The client-side application communicates with the server-side service through a service reference. The service reference provides the client-side application with the interface to the DownloadService class, allowing it to call its methods.

A black screen with white text

Description automatically generated

DownloadService is a class that implements the IDownloadService interface. This interface defines a contract for the services that Download Service provides.



A screen shot of a computer program

Description automatically generated

1. Designing the Data

The project uses a DownloadDetails class to store details about each download, such as progress, total file size, bytes copied, start time, pause time, and a cancellation token source. This data is used to track the status of each download and control its operation.

A screen shot of a computer program

Description automatically generated

A computer screen with text

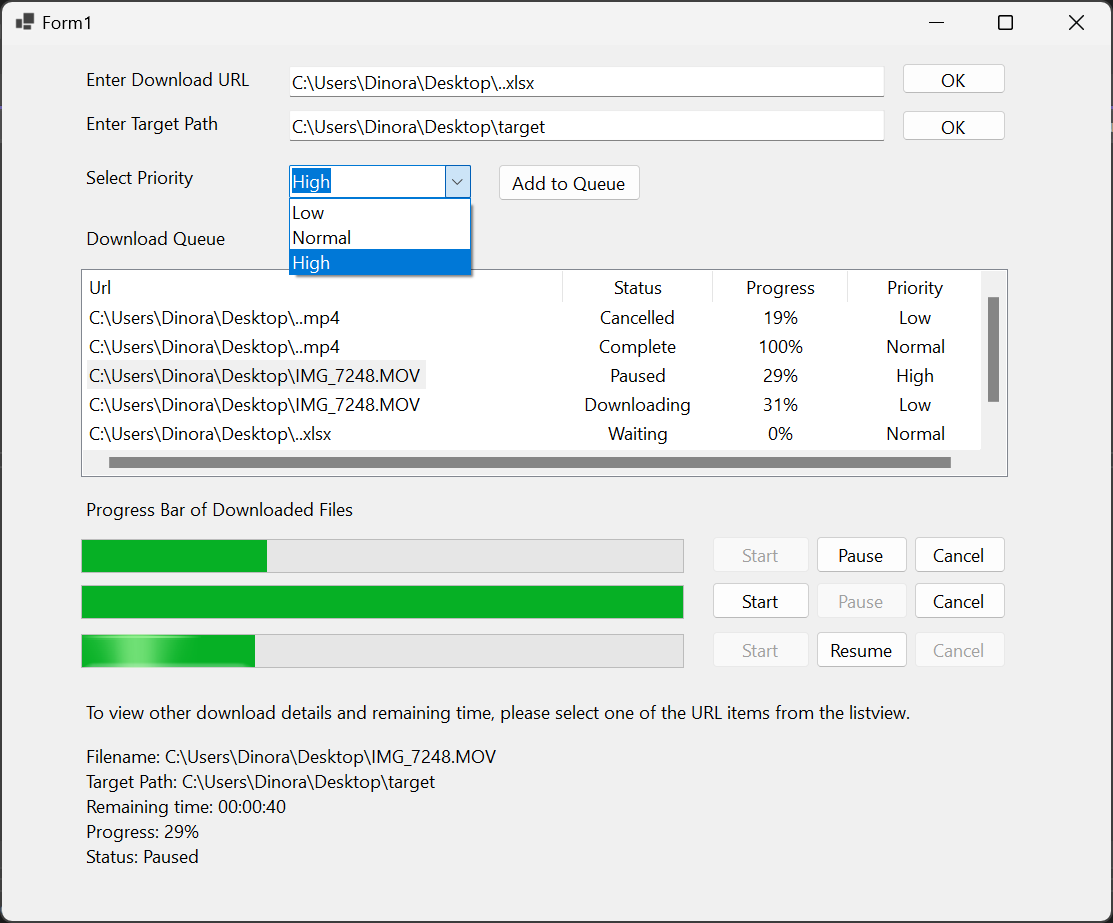
Description automatically generated

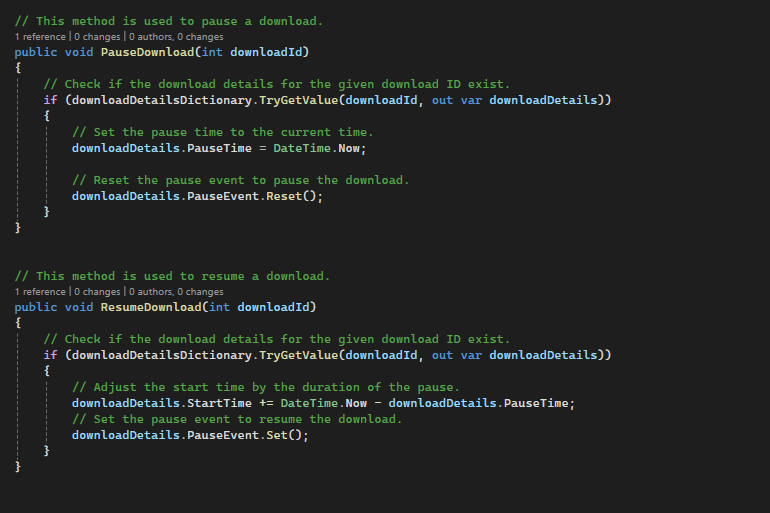
**Implementation**

**Display Screen (Windows Forms Client)**

The client application offers a user-friendly interface for initiating and managing downloads. Input sections allow users to enter download URLs and customize download settings such as priority and target path. A ListView displays queued URLs along with their status, progress, and priority, providing users with an overview of ongoing downloads. Additionally, buttons are provided for starting, pausing/resuming, and canceling individual downloads. Progress bars and labels provide visual feedback on download progress, enhancing the user experience and usability of the application.

Pause/resume downloads



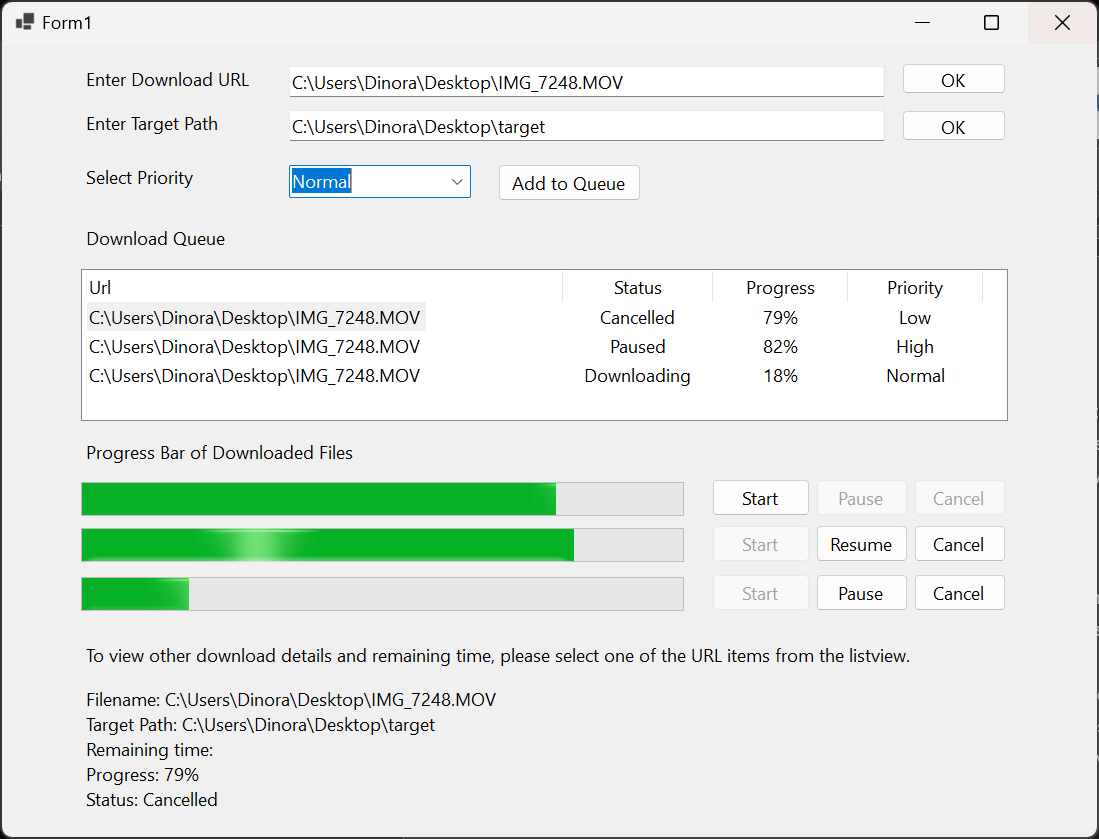
Server-side code: 

Pause/resume buttons code

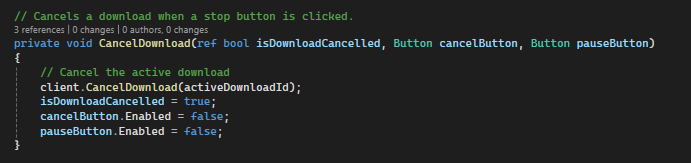
A computer screen shot of a program code

Description automatically generated

Cancel download:



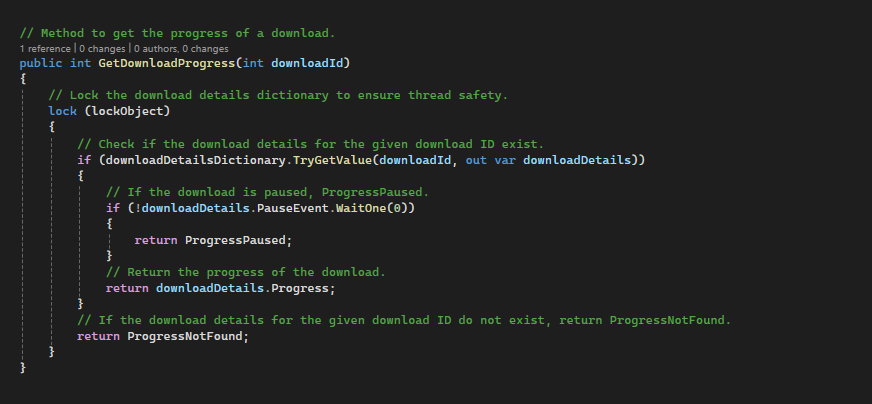
When a download is canceled, the status is automatically updated, and the cancel button is disabled to prevent accidental double cancels. Additionally, the start button activates to start a new download.



A computer screen shot of a program

Description automatically generated

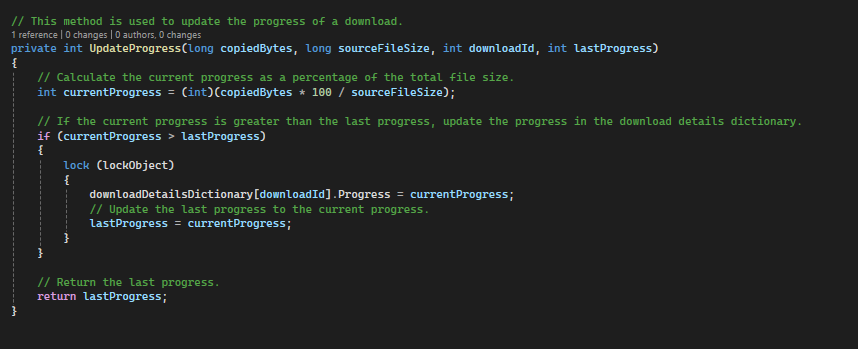
Get downloaded progress in server side:



Get remaining time:

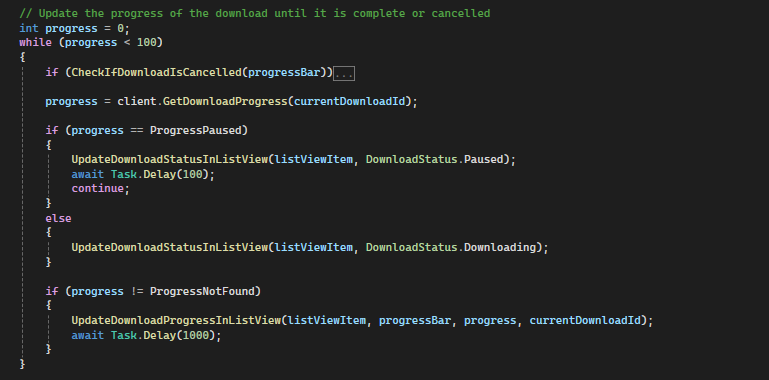
A computer screen shot of a program code

Description automatically generated

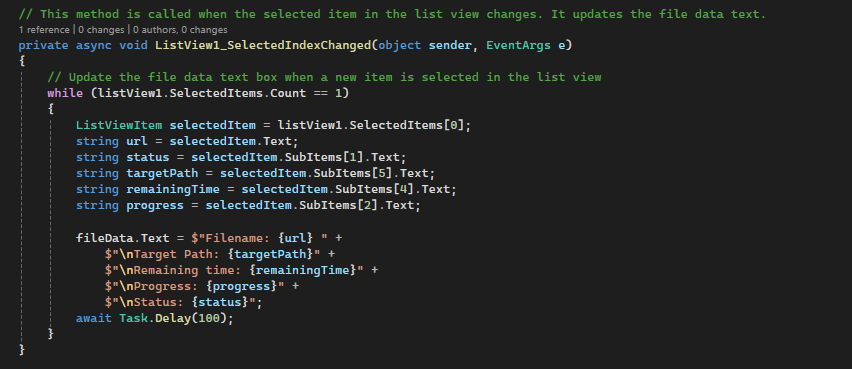
Update progress: 

A computer screen with text on it

Description automatically generated



This method shows details of the downloaded file, when one of the items in list view is selected:



Inputs validation:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen with text

Description automatically generated

A screenshot of a computer

Description automatically generated

A black screen with colorful text

Description automatically generated

Server-Side Service (DownloadService.cs)

This is where the functionality of the download service is implemented. It includes methods for downloading files, pausing, resuming, and canceling downloads, getting the download progress and remaining time. It uses a DownloadDetails class to store details about each download.

Service Interface (IDownloadService.cs)

This is where the interface for the download service is defined. It includes the definitions of the methods provided by the service. The client-side application uses this interface to interact with the server-side service.

Form Design (Form1.Designer.cs)

This is where the design of the form is implemented. It includes the initialization of the various controls on the form and their properties.

Integration of Client and Server

The client-side application communicates with the server-side service through a service reference. The service reference provides the client-side application with the interface to the DownloadService class, allowing it to call its methods.

**Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test case** | **Description** | **Input** | **Actual result** |
| 1 | Download file | Provide a valid download URL, target path to the text box. | The method returns true, and the file is uploaded to the specified target path. |
| 2 | Pause and resume download | * Start a download, then click to pause button in the form. * Click to resume button in the form. | * The download process is paused. * The download process is resumed |
| 3 | Cancel download | Start a download, then click to cancel button in the form. | The download process is canceled. |
| 4 | Get download progress | Start a download, user views the progress of downloaded file in the form. | Return the current progress of the download. |
| 5 | Get remaining time | Start a download, user views the remain time of the download file in the form. | Return the estimated remaining time for the download. |

1. **Deployment**

In the deployment phase of the Software Development Life Cycle (SDLC), the application is made available for use in a production environment (Upadhyay, 2021).

For this download manager project, it could be deployed on an Internet Information Services (IIS) server. This would allow the server-side service (Download Service.cs) to be hosted on the IIS server, making it accessible over the network.

The client-side application (Form1.cs) could then send requests to the server-side service over the network, allowing it to download files stored within the website's content directory on the IIS server.

Downloading Service (WCF):

The DownloadService, implemented as a WCF service, serves as the backbone of the system.

Conclusion

In conclusion, the download manager project offers a comprehensive solution for efficient file downloading, consisting of a Web Service and a Windows Forms Client Application. The design of the project is based on a client-server architecture that ensures scalability. Various concurrency techniques, including multithreading, locks, ManualResetEvent, CancellationToken, and Semaphore, have been employed to enhance performance and responsiveness while eliminating potential problems like race conditions and deadlocks. The project goes through the stages of the Software Development Life Cycle (SDLC), from requirements gathering and analysis to deployment, providing a systematic and well-structured development process. Through careful implementation and testing, the download manager project stands ready for deployment, offering an intuitive user interface and reliable download functionality that meets the requirements of the modern digital world.

Reference

Asryan, D. (no date). A .NET Programmer’s Guide to CancellationToken | Toptal®. *Toptal Engineering Blog*. Available from https://www.toptal.com/asp-dot-net/dotnet-programmer-guide-to-cancellationtoken [Accessed 3 April 2024].

asyncio - Synchronization Primitives in Concurrent Programming using Async / Await Syntax by Sunny Solanki. (2021). *coderzcolumn.com*. Available from https://coderzcolumn.com/tutorials/python/asyncio-synchronization-primitives-in-concurrent-programming-using-async-await-syntax [Accessed 1 April 2024].

BillWagner. (2022). Overview of synchronization primitives. *learn.microsoft.com*. Available from https://learn.microsoft.com/en-us/dotnet/standard/threading/overview-of-synchronization-primitives.

Clark, H. (2022). 6 Stages Of The Software Development Life Cycle (SDLC). *The Product Manager*. Available from https://theproductmanager.com/topics/software-development-life-cycle/.

Concurrency in Operating System. (no date). *www.tutorialspoint.com*. Available from https://www.tutorialspoint.com/concurrency-in-operating-system.

Concurrency vs Parallelism: The Differences- NetNut. (2023). *netnut.io*. Available from https://netnut.io/concurrency-vs-parallelism/?utm\_medium=organic&utm\_source=twitter [Accessed 2 April 2024].

Controlling thread order using semaphores in C. (no date). *Stack Overflow*. Available from https://stackoverflow.com/questions/41143974/controlling-thread-order-using-semaphores-in-c [Accessed 2 April 2024].

Deadlock in Java Multithreading - GeeksforGeeks. (2017). *GeeksforGeeks*. Available from https://www.geeksforgeeks.org/deadlock-in-java-multithreading/.

Difference in usage and implementation of ManualResetEvent(Slim), Semaphore(Slim) and ReaderWriterLock(Slim). (no date). *Stack Overflow*. Available from https://stackoverflow.com/questions/6764864/difference-in-usage-and-implementation-of-manualreseteventslim-semaphoreslim [Accessed 1 April 2024].

dotnet-bot. (no date). Semaphore Class (System.Threading). *learn.microsoft.com*. Available from https://learn.microsoft.com/en-us/dotnet/api/system.threading.semaphore?view=net-8.0 [Accessed 3 April 2024].

IEvangelist. (2022). Task Cancellation - .NET. *learn.microsoft.com*. Available from https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/task-cancellation [Accessed 3 April 2024].

Java Concurrency: Master the Art of Multithreading. (2023). *BairesDev Blog: Insights on Software Development & Tech Talent*. Available from https://www.bairesdev.com/blog/java-concurrency/.

MSFT, S.T. -. (2012). Building Async Coordination Primitives, Part 1: AsyncManualResetEvent. *.NET Parallel Programming*. Available from https://devblogs.microsoft.com/pfxteam/building-async-coordination-primitives-part-1-asyncmanualresetevent/ [Accessed 1 April 2024].

OS Processes, Threads, and Semaphores - DZone. (no date). *dzone.com*. Available from https://dzone.com/articles/os-processes-threads-and-semaphores [Accessed 2 April 2024].

Pinho, L. (no date). *Real-Time Parallel Programming: State of Play and Open Issues*. Available from https://arxiv.org/ftp/arxiv/papers/2303/2303.11018.pdf [Accessed 2 April 2024].

Possible Race condition with ManualResetEvent. (no date). *Stack Overflow*. Available from https://stackoverflow.com/questions/16122051/possible-race-condition-with-manualresetevent [Accessed 29 March 2024].

Semaphores and its types. (2021). *GeeksforGeeks*. Available from https://www.geeksforgeeks.org/semaphores-and-its-types/.

*Synchronization Primitives mutex, semaphore, condition variable*. (no date).

Upadhyay, D. (2021). SDLC Deployment Phase - A Step by Step Guide [2021]. *OpenXcell*. Available from https://www.openxcell.com/blog/sdlc-deployment-phase/.

Uploading and Downloading: What It Means. (2023). *CitizenSide*. Available from https://citizenside.com/technology/uploading-and-downloading-what-it-means/ [Accessed 2 April 2024].

Visual Paradigm Online. (no date). *online.visual-paradigm.com*. Available from https://online.visual-paradigm.com/drive/#infoart:proj=0&dashboard.