

Multiple clients support: introducing programming threads

General description

This assignment focuses on a crucial aspect of software development: handling multiple clients concurrently through the introduction of programming threads. You will learn how to use threading to enable effective communication between a server and numerous clients simultaneously. Through practical exercises, you'll gain proficiency in thread management and addressing **basic** challenges associated with concurrent programming.

Task

It is strictly prohibited to use AI and other tools for generating the solution for your assignment! Your goal is to improve your programming skills, that's why please concentrate on learning instead of cheating.

Upgrade the client-server application from the assignment 1 to supports multiple concurrent the same time.

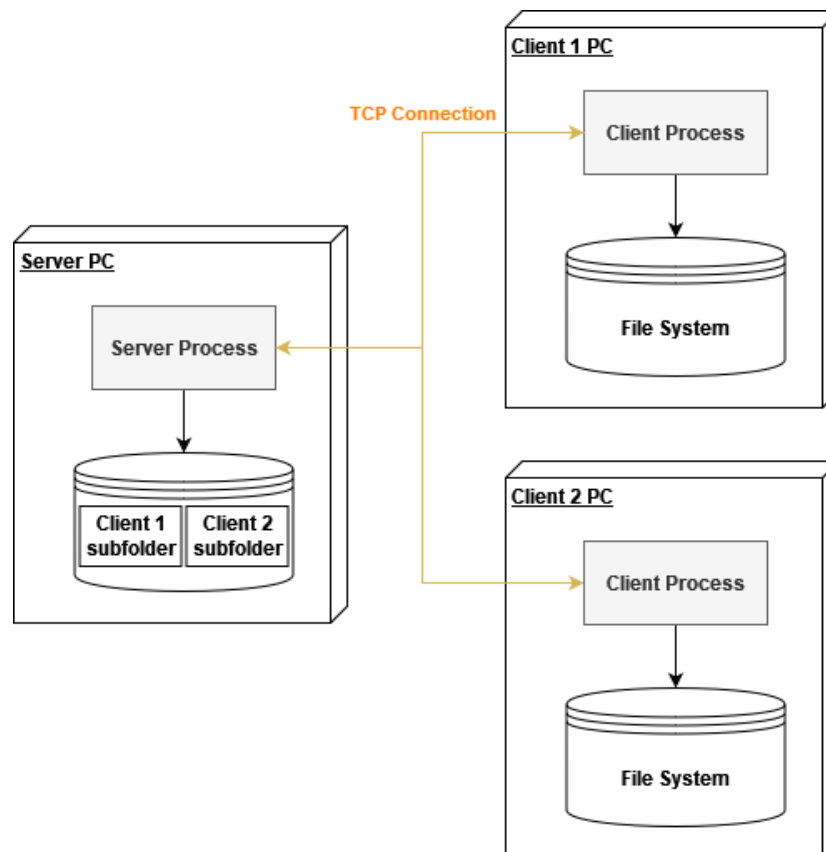


Fig. 1. Deployment view of a client-server architecture with multiple clients

Server Implementation:

- Implement a server program capable of accepting multiple client connections simultaneously.
- Each client should work with its own subfolder on the server.
- Utilize threading to handle multiple client requests concurrently.

Client Implementation:

- Implement a client program that connects to the server and does the same set of actions as in the assignment 1.
- During the connection the client can provide its name to the server to create/choose a folder on the server's side.
- Implement error handling to manage situations where the server is busy or unresponsive.

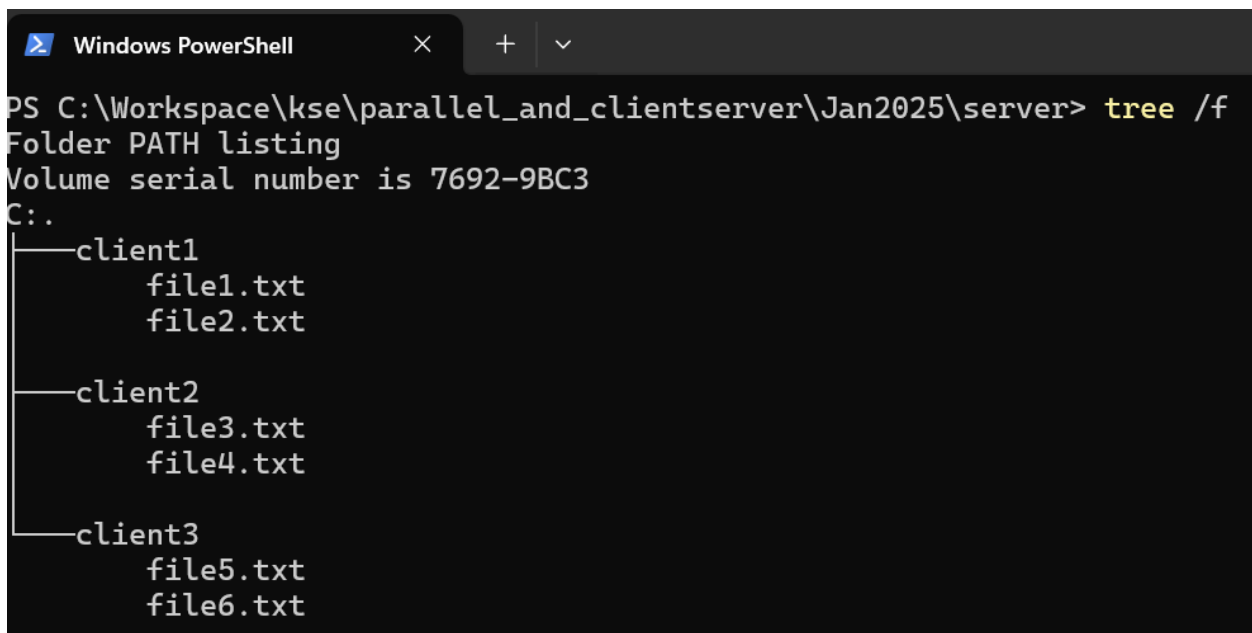
Communication Protocol:

- Establish a clear communication protocol between the server and clients. Define the format of messages for requesting operations and receiving results.

Updated commands:

- Each client should add its name at the beginning, e.g. **GET <CLIENT_NAME> <FILE_NAME>**

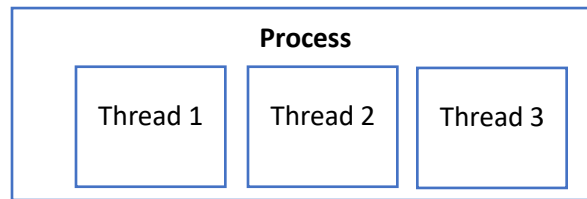
File tree view on the server:



```
Windows PowerShell
PS C:\Workspace\kse\parallel_and_clientserver\Jan2025\server> tree /f
Folder PATH listing
Volume serial number is 7692-9BC3
C:.\
├── client1
│   ├── file1.txt
│   └── file2.txt
├── client2
│   ├── file3.txt
│   └── file4.txt
└── client3
    ├── file5.txt
    └── file6.txt
```

Fig. 2. Filesystem on the server side

Theory



- **Process:**
 - A process is an independent program with its own memory space, resources, and state.
 - Processes are isolated from each other and communicate through inter-process communication (IPC) mechanisms.
- **Thread:**
 - A thread is a lightweight unit of execution within a process.
 - A thread is a subsequent set of instructions that will be executed on CPU.
 - Threads within the same process share the same memory space, allowing them to communicate easily.

Handling Multiple Clients:

- Each client connection can be handled **by a separate thread**, allowing the server to serve multiple clients simultaneously.
- The main Server's thread will be listening for new connections and after accepting it a new thread will be created and the server will pass a socket handle to that thread.

```
#include <iostream>
#include <thread>
#include <vector>

const int numThreads = 4; // Number of threads
const int arraySize = 100; // Size of the array

// Function that calculates the sum of elements in a portion of the array
void calculateSum(const std::vector<int>& array, int start, int end, int& partialSum) {
    partialSum = 0;
    for (int i = start; i < end; ++i) {
        partialSum += array[i];
    }
}

int main() {
    std::vector<int> myArray(arraySize, 1); // Initialize array with 1s

    // Calculate the portion size each thread will handle
    int portionSize = arraySize / numThreads;

    // Container to store thread objects
    std::vector<std::thread> threads;
```

```

// Container to store partial sums
std::vector<int> partialSums(numThreads, 0);

// Create threads and assign each thread a portion of the array
for (int i = 0; i < numThreads; ++i) {
    int start = i * portionSize;
    int end = (i == numThreads - 1) ? arraySize : (i + 1) * portionSize;

    threads.emplace_back(calculateSum, std::ref(myArray), start, end, std::ref(partialSums[i]));
}

// Join the threads to wait for their completion
for (auto& thread : threads) {
    thread.join();
}

// Calculate the final sum by summing up the partial sums
int finalSum = 0;
for (int partialSum : partialSums) {
    finalSum += partialSum;
}

std::cout << "Sum of array elements: " << finalSum << std::endl;

return 0;
}

```

In the above example a simple multithreaded program has been developed:

- The calculateSum function calculates the sum of elements in a portion of the array. Each thread will call this function to calculate its portion of the sum.
- The main function initializes an array with 100 elements, each set to 1.
- The array is divided into portions, and each thread is responsible for calculating the sum of its portion.
- Threads are created and assigned portions of the array to process. The results are stored in partialSums.
- The main thread waits for all the threads to finish using the join function.
- The final sum is calculated by summing up the partial sums.

Attention! This assignment does not require to support synchronization, however please use <mutex> library if you want to have some synchronized parts. It's recommended to contact lecturer and explain the need of synchronization.

Examples

The key difference between assignments 1 and 2 is using the below part of the code to support multiple clients at the same time in the concurrent threads:

```
#include <thread>
void handleClient(int clientSocket) {
    ...
}
...
// Vector to store thread objects
std::vector<std::thread> threads;

// Accept and handle incoming connections
while (true) {
    // Accept a client connection
    int clientSocket = accept(serverSocket, nullptr, nullptr);
    if (clientSocket == -1) {
        std::cerr << "Error accepting client connection: " << strerror(errno) <<
std::endl;
        continue;
    }

    // Create a new thread to handle the client
    threads.emplace_back(handleClient, clientSocket);
}

// Join threads to wait for their completion (this won't be reached in this
example)
for (auto& thread : threads) {
    thread.join();
}
```

Evaluation

Attention! The grade will be decreased in case you can't explain how the code works.

Client-server implementation; the server should support multiple clients at the same time	8
Communication protocol description documentation of the project internals (including at least one UML sequence diagram with both server and client mentioned)	2
Extra: server should be backward compatible with clients from the assignment #3	1
Extra: statistics calculation on the server side, how many times each command is called	1
Total	10 + 2(extra)

Absence of Git will lead to 0 points. <https://classroom.github.com/a/TkCNRcMi>
Incorrect answers on theoretical questions will lead to 0 points.

Questions

1. What is a thread?
2. What is the difference comparing a thread and a process?
3. How can you create threads?
4. What is a multithreaded server, and how does it differ from a single-threaded server?
5. Explain the terms "concurrency" and "parallelism" in the context of multithreading.
6. Why might a developer choose to implement a multithreaded server instead of a single-threaded server?
7. How do threads in a multithreaded server share resources such as memory? How about thread's stack?
8. How to share data between threads?
9. Which issues could occur in a multithreaded environment?
10. Identify areas in your implementation where additional error handling could be added.

Links

1. Internetworking With TCP/IP by Douglas E. Comer :
[https://www.homeworkforyou.com/static_media/uploadedfiles/Douglas%20E.%20Comer%20-%20Internetworking%20with%20TCP IP%20Volume%20One.%201-Addison-Wesley%20\(2013\).pdf](https://www.homeworkforyou.com/static_media/uploadedfiles/Douglas%20E.%20Comer%20-%20Internetworking%20with%20TCP%20IP%20Volume%20One.%201-Addison-Wesley%20(2013).pdf)
2. Computer Systems: A Programming Perspective: [https://github.com/iWangMu/Book-CSAPP/blob/master/ Attachments/Computer Systems A Programmers Perspective\(3rd\).pdf](https://github.com/iWangMu/Book-CSAPP/blob/master/Attachments/Computer%20Systems%20A%20Programmers%20Perspective(3rd).pdf) Chapters 11-12