CT30A3401
Distributed Systems
Lecture 4

Bilal Naqvi, PhD.
syed.naqvi@lut.fi

Processes and Threads

- A process is often defined as a program in execution
  - example: a program that is currently being executed on one of the operating system's virtual processors
- Thread is the segment of a process
  - a process can have multiple threads and these multiple threads are contained within a process
  - example: a threaded web-server

Processes vs Threads

Process	Thread
takes more time for creation and termination	takes less time for creation and termination
takes more time for context switching	takes less time for context switching
less efficient in term of communication.	more efficient in term of communication
consume more resources	consume less resources
isolated	share memory

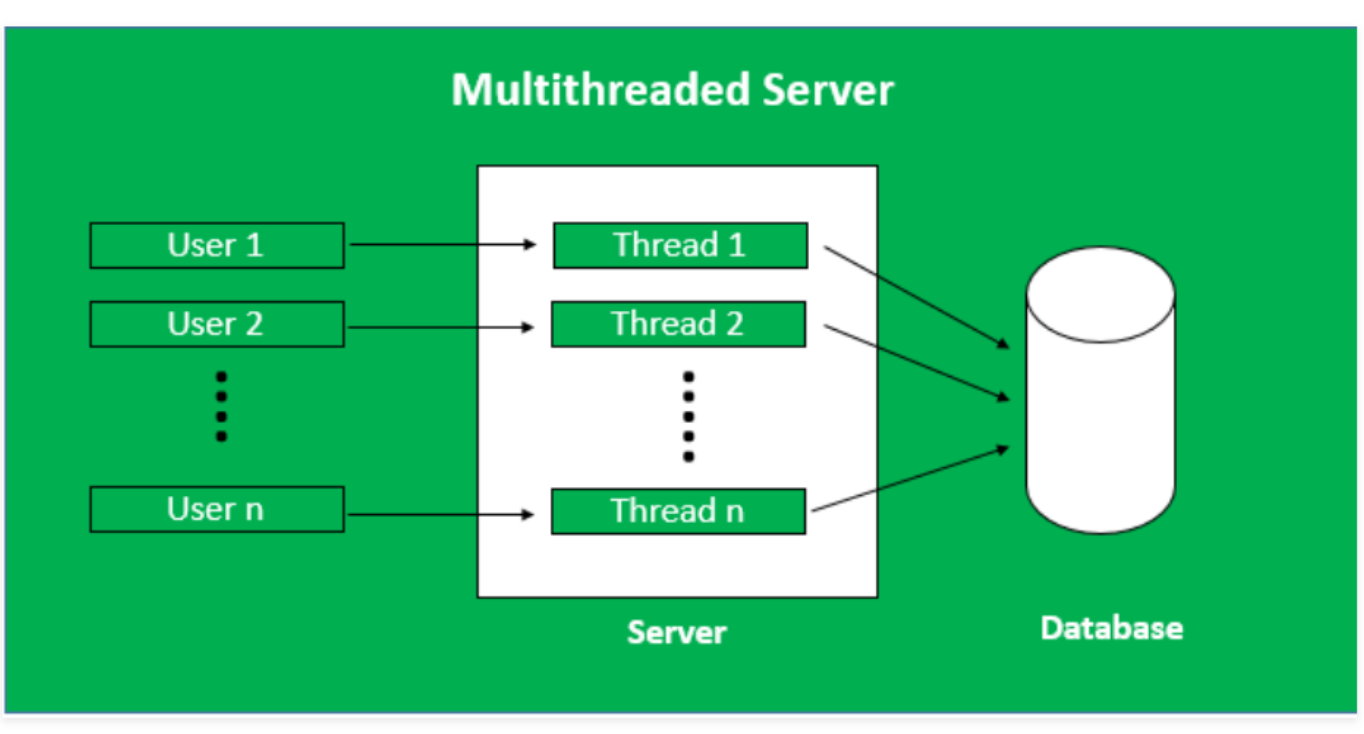
Processes vs Threads (continued)

Process	Thread
process switching uses interface in the operating system	thread switching does not require to call a operating system and cause an interrupt to the kernel
if one server process is blocked no other server process can execute until the first process unblocked	second thread in the same task could run, while one server thread is blocked
process has its own Process Control Block, Stack and Address Space	thread has parents' PCB, its own Thread Control Block and Stack and common Address space

Threads in Distributed Systems

- Multithreaded clients/servers
- A server having more than one thread is known as Multithreaded Server
  - when a client sends the request, a thread is generated through which a user can communicate with the server
  - need to generate multiple threads to accept multiple requests from multiple clients at the same time

Multithreaded Server

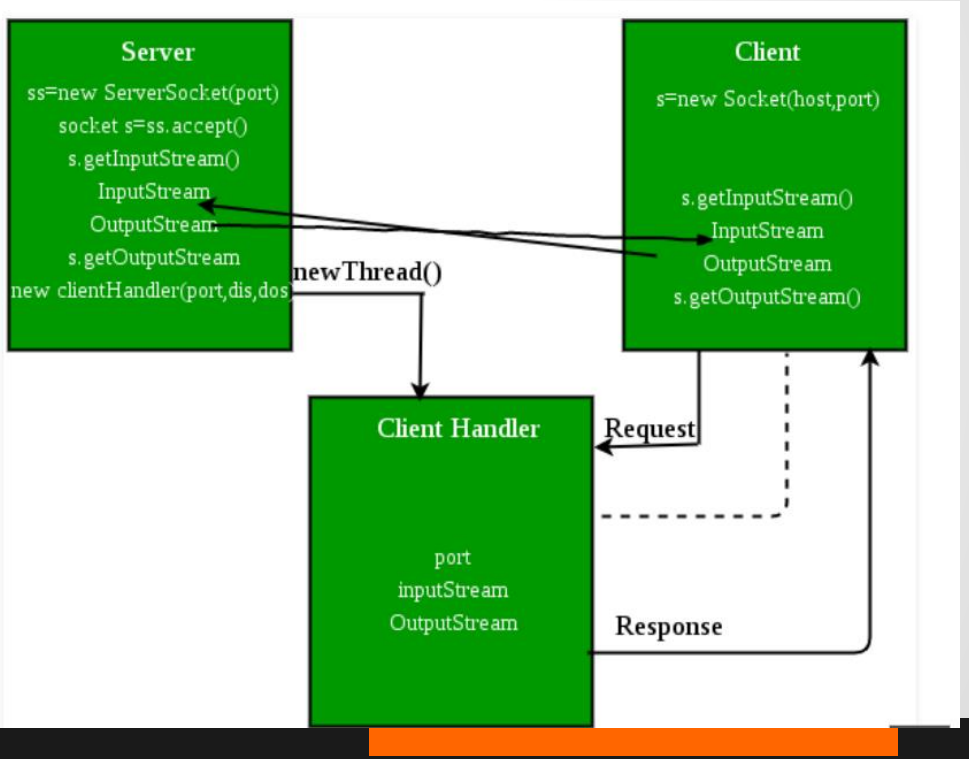


Multithreaded Servers

- Merits
  - quick and efficient
  - waiting time for users decreases
  - threads are independent of each other
  - issue in one thread does not affect other threads
- Demerits
  - complicated code
  - debugging is difficult

How it works

- Client file contains only one class Client (for creating a client)
- Server file has two classes, Server(creates a server) and ClientHandler(handles clients using multithreading)



Client-side program

- Client-Side Program does two things
  - establish a Socket Connection
  - communication

```java
import java.io.*;
import java.net.*;
import java.util.*;

// Client class
class Client {

    // driver code
    public static void main(String[] args)
    {
        // establish a connection by providing host and port
        // number
        try (Socket socket = new Socket("localhost", 1234)) {

            // writing to server
            PrintWriter out = new PrintWriter(
                socket.getOutputStream(), true);

            // reading from server
            BufferedReader in
                = new BufferedReader(new InputStreamReader(
                    socket.getInputStream()));

            // object of scanner class
            Scanner sc = new Scanner(System.in);
            String line = null;

            while (!"exit".equalsIgnoreCase(line)) {

                // reading from user
                line = sc.nextLine();

                // sending the user input to server
                out.println(line);
                out.flush();

                // displaying server reply
                System.out.println("Server replied "
                                   + in.readLine());
            }

            // closing the scanner object
            sc.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client-side program example

Server-side program

- **Server**
  - establishing the connection: server socket object is initialized and inside a while loop a socket object continuously accepts an incoming connection
  - obtaining the streams: the inputstream object and outputstream object is extracted from the current requests' socket object.
  - creating a handler object: After obtaining the streams and port number, a new clientHandler object (the above class) is created with these parameters.
  - invoking the start() method: The start() method is invoked on this newly created thread object.

Server-side program

- **ClientHandler**
  - First, this class implements Runnable interface so that it can be passed as a Runnable target while creating a new Thread
  - Secondly, the constructor of this class takes a parameter, which can uniquely identify any incoming request, i.e. a Socket
  - Inside the run() method of this class, it reads the client's message and replies

```java
// Server class
class Server {
    public static void main(String[] args)
    {
        ServerSocket server = null;

        try {

            // server is listening on port 1234
            server = new ServerSocket(1234);
            server.setReuseAddress(true);

            // running infinite loop for getting
            // client request
            while (true) {

                // socket object to receive incoming client
                // requests
                Socket client = server.accept();

                // Displaying that new client is connected
                // to server
                System.out.println("New client connected"
                                   + client.getInetAddress()
                                         .getHostAddress());

                // create a new thread object
                ClientHandler clientSock
                    = new ClientHandler(client);

                // This thread will handle the client
                // separately
                new Thread(clientSock).start();
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            if (server != null) {
                try {
                    server.close();
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

```java
// ClientHandler class
private static class ClientHandler implements Runnable {
    private final Socket clientSocket;

    // Constructor
    public ClientHandler(Socket socket)
    {
        this.clientSocket = socket;
    }

    public void run()
    {
        PrintWriter out = null;
        BufferedReader in = null;
        try {

              // get the outputstream of client
            out = new PrintWriter(
                clientSocket.getOutputStream(), true);

              // get the inputstream of client
            in = new BufferedReader(
                new InputStreamReader(
                    clientSocket.getInputStream()));

            String line;
            while ((line = in.readLine()) != null) {

                // writing the received message from
                // client
                System.out.printf(
                    " Sent from the client: %s\n",
                    line);
                out.println(line);
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            try {
                if (out != null) {
                    out.close();
                }
                if (in != null) {
                    in.close();
                    clientSocket.close();
                }
            }
            catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```