Birzeit University
Faculty of Engineering and Technology
Department of Electrical and Computer Engineering
ENCS3320 – Computer Networks (Term 1242)
Project #1 (*Socket Programming*) – Due Thursday, May 08, 2025

A) Objectives

This project aims to deepen your understanding of socket programming and computer networking. Specifically, the objectives are as follows:

- Gain familiarity with fundamental network commands.
- Understand how to create Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) sockets.
- Learn the basics of HyperText Markup Language (HTML) and web server configuration.
- Develop teamwork skills by collaborating on project tasks.

B) Requirements and Deliverables

This project includes ***three tasks***. As a team of three students (*from any section*), you are responsible for completing **all** tasks and submitting both your source codes and a project report. Please consider the following requirements:

- **Implementation Guidelines**: For **Tasks 2** and **3**, you may choose *any programming language* (e.g., Python, Java, C, etc.). Note that libraries should not be used for implementing socket functionalities; *only the standard socket library is allowed for task 2*. The implementation should reflect *your own work* and be adaptable for future expansion with minimal effort (i.e., your implementation should be as generic as possible).
- **Team Coordination**: Each team member should actively participate across all project stages, including research, design, implementation, testing, and report preparation. To support collaboration, we recommend using GitHub for version control and collaborative coding, and Overleaf for seamless, real-time document editing. Consider dividing responsibilities clearly, sharing regular feedback, and tracking progress closely to ensure a unified and successful project outcome.

Please submit the following **files** as *a single compressed folder* (**.zip**) named ***TeamName_Project1.zip*** (TeamName is given in the excel sheet, e.g., team name for group#1 is **T001 → T001_Project1.zip**) through **ITC** (https://itc.birzeit.edu/course/view.php?id=34254):

1) **Project Report** (***TeamName***_*report.pdf*): A detailed report in PDF format, documenting your solutions to each task. See ***Section D (Report and Grading Criteria)*** for further guidelines.
2) **Task 2 Folder**: A folder named ***Task2*** containing well-documented source code and files for Task 2. This folder should include: (i) The main server code (e.g., *server.py*) and (ii) Subfolders named *html*, *css*, and *imgs* containing the necessary HTML, CSS, and image files for the server, respectively. Include as many subfolders as needed to make your implementation well organized.
3) **Task 3 Folder**: A folder named ***Task3*** with well-documented source code for running the server (e.g., *server.py*) and client (e.g., *client.py*) for Task 3.

Each team should submit one final version. The submission deadline is end of **May 08, 2025**.

C) Project Tasks

- **Task 1 – Network Commands and Wireshark:**
  A) Briefly explain each of the following commands in your own words, using no more than two sentences per command: (i) **ipconfig**, (ii) **ping**, (iii) **tracert/traceroute**, (iv) **telnet**, and (v) **nslookup**.
  B) Make sure your computer is connected to the internet, then complete the following steps:
    1. Execute the **ipconfig /all** command on your computer and locate the IP address, subnet mask, default gateway, and DNS server addresses for your main network interface.
    2. Send a **ping** request to a device within your local network, such as from a laptop to a smartphone connected to the same wireless network.
    3. **Ping** gaia.cs.umass.edu, and using the results, provide a brief explanation of whether you believe the response originates from.
    4. Run **tracert/traceroute** gaia.cs.umass.edu to see the route it takes.
    5. Execute the **nslookup** command to retrieve the Domain Name System (DNS) information for gaia.cs.umass.edu.
    6. Use **telnet** to try connecting to gaia.cs.umass.edu at port 80.
    7. Provide details about the autonomous system (AS) number, number of IP addresses, prefixes, peers, and the name of Tier 1 ISP(s) associated with gaia.cs.umass.edu. Hint, you can use any online tool (e.g., www.bgpview.io or https://bgp.tools/).
  C) Use **Wireshark** to capture a DNS query and response for gaia.cs.umass.edu (make sure that you clear your cache before request this website).
  Wireshark can be downloaded from https://www.wireshark.org/download.html.

- **Task 2 – Implement a Simple Web Server Using Socket Programming**

Implement a simple, complete web server using socket programming that listens on a port number derived from the student ID of one team member. For example, if the student ID is 1201**X**1**Y**, the server should listen on port **99XY** (e.g. 1201516 → Port is 9956). Ensure the code is written to be as general and reusable as possible. The server should support the following HTML web pages:

a) Main English Webpage (main_en.html):
This page should include:
- The text "**ENCS3320-Webserver**" displayed on the web browser tab in RED color.
- The title "**Welcome to ENCS3320 - Computer Networks Webserver**" at the top of the page in BLUE color.
- Team members' photos (in **.png** format) displayed alongside their names and IDs, arranged in a table-like structure. Organize each member's information within separate boxes on the page.
- A brief description of each team member, covering details like projects completed in various courses, skills, hobbies, etc.
- A clearly organized presentation of a topic from the first chapter of the textbook, incorporating elements like paragraphs, images (in **.jpg** format), and both ordered and unordered lists. Ensure the use of suitable headings, font styles, colors, and text formatting (bold/italic).
- Use CSS to make the page looks nice.
- Links to external resources, including:
  1. Textbook website (https://gaia.cs.umass.edu/kurose_ross/index.php).
  2. Birzeit website (https://www.birzeit.edu/).
- A link to a local html file (named by *mySite_**STDID**_en.html* → *mySite_**1201516**_en.html*), as described below.

b) Local Webpage (mySite_STDID_en.html):
This page should include a form that allows users to request an image or video related to the computer networking topic presented in (**main_en.html**) by entering the file name into an input field. If the requested file is not available, the server should respond with a "**307 Temporary Redirect**" status code, redirecting the client (web browser) based on the type of file requested:

- Images: Redirect to a **Google** search URL that filters results to images based on the user's input.
- Videos: Redirect to a **Google** search URL that filters results to videos based on the user's input.

c) Arabic Version (main_ar.html):
Provide Arabic versions of the **main_en.html** and **mySite_STDID_en.html** files, named **main_ar.html** and **mySite_STDID_ar.html**, respectively.

d) Server Functionality:
When the server receives an HTTP request, it should:

- Print the HTTP request details to the terminal.
- Generate and send an HTTP response that includes the requested content and specifies the correct Content-Type based on the file type (e.g., text/html for HTML, text/css for CSS, image/png for PNG, and video/mp4 for MP4).
- Response codes (e.g., 200 OK, 404 Not Found) should be logged in the server terminal, along with the client's IP address and port number, as well as the server's port number.

e) Default and Error Responses:

- For requests targeting **/**, **/en**, **/index.html**, or **/main_en.html** (e.g., http://localhost:99XY/, http://localhost:99XY/en, http://localhost:99XY/index.html, or http://localhost:99XY/main_en.html), the server should respond with **main_en.html**.
- For requests targeting **/ar** or **/main_ar.html** (e.g., http://localhost:99XY/ar or http://localhost:99XY/main_ar.html), the server should respond with **main_ar.html**.
- If the client requests an invalid URL (i.e., a file not found on the server), the server should return a simple HTML error page with:
  - The status line "**HTTP/1.1 404 Not Found**"
  - The text "**Error 404**" in the browser tab
  - The message "**The file is not found**" displayed in red text in the body
  - The client's IP address and port number

Organize the server files as outlined in ***Section B (Requirements and Deliverables)***.

• Task 3 – TCP/UDP Hybrid Client-Server Game Using Socket Programming

In this task, you are required to develop a hybrid networked guessing game where a server uses TCP for control messages (e.g., game setup, scoring) and UDP for real-time gameplay (e.g., player updates). The game should support multiple players competing in a simple real-time environment. The server manages the overall game flow, which is structured into multiple rounds. Below are the descriptions about the game features, components and protocols used, game workflow of both the server and clients, along with the expected terminal outputs for each, and initial setting of the game.

a) Game Features:

This is a multiplayer guessing game where players (i.e. clients) compete to guess a secret number generated by the server. The game uses a hybrid approach with TCP for initial connection and game setup, and UDP for fast-paced guessing rounds to minimize latency. Here are the main features of this game:

- Players compete to guess a random (i.e. secret) number generated by the server.
- The game is round-based and allows multiple players (2 or more) simultaneously.
- Players register via TCP and submit guesses via UDP for speed. It is a hybrid protocol usage (TCP for reliable setup, UDP for fast guesses).

- Real-time feedback on guesses, where the server responds with feedback ("Higher", "Lower", or "Correct") via UDP.
- When a player wins, results are sent to all players via TCP.

b) Components and Protocol Design:

This game operates in two main phases: one for establishing the connection and another for handling updates. Here are more details about the two phases:

1. **TCP Phase** (Reliable Connection Setup):
   a) Client connects to server on known TCP port
   b) Server sends welcome message and requests player name
   c) Client sends player name
   d) If the "player name" is unique the Server acknowledges and waits for minimum players, otherwise, the server sends "use another player name"
   e) When enough players connect, server sends game start message with rules
   f) Once a player guesses the correct value, the server announces the winner and prompts to start a new round. If multiple players guess correctly, the result is announced, and a new game begins.
2. **UDP Phase** (Fast-Paced Guessing):
   a) Server generates random number in pre-defined range
   b) Server broadcasts round start with range info to all players
   c) Players send guesses via UDP to server's UDP port
   d) Server processes guesses, determines proximity to secret number
   e) Server sends feedback to each player about their guess
   f) First player to guess correctly wins the round

c) Game Workflow:

1. **Server Initialization:**
   a) Server opens TCP socket on port X and UDP socket on port Y.
   b) Waits for players to connect and register via TCP.
2. **Player Registration (TCP):**
   a) Clients connect to TCP server.
   b) Send JOIN <username> command.
   c) Server stores names and sends game instructions.
3. **Game Start:**
   a) When minimum players (e.g., 2) have joined, server broadcasts "Game started with …" (as listed in the cases below) via TCP.
   b) Random number is chosen (e.g., 1 to 100).
4. **Guessing Phase (UDP):**
   a) Clients send guesses to the server via UDP.
   b) Server responds with:
   "Higher"
   "Lower"
   "Correct"
5. **Victory Condition:**
   a) First player to guess correctly is the winner.
   b) Server sends final results to all clients via TCP.

d) Test Cases and Expected Output:

Below are three cases that you should test and include them in your report, along with the expected output for each:

1. Two players register, game starts, one wins:

Server, Player 1, and Player 2 terminals will appear as follows:

Server Terminal

```
Server started on <server_host>: TCP 6000, UDP 6001
New connection from ('192.168.1.5', 54321) as player_1
New connection from ('192.168.1.6', 54322) as player_2
Starting game with 2 players...
Game completed. Winner: player_1
```

Player_1 Terminal

```
Connected as player_1
UDP connection established

Waiting Room:
player_1
player_2

Game started with players: player_1, player_2
You have 60 seconds to guess the number (1-100)!
Enter your guess (1-100): 50
Feedback: Lower
Enter your guess (1-100): 60
Feedback: Higher
Enter your guess (1-100): 63
Feedback: CORRECT

=== GAME RESULTS ===
Target number was: 63
Winner: player_1
```

Player_2 Terminal

```
Connected as player_2
UDP connection established

Waiting Room:
player_1
player_2

Game started with players: player_1, player_2
You have 60 seconds to guess the number (1-100)!
Enter your guess (1-100): 70
Feedback: Higher
Enter your guess (1-100): 50
Feedback: Lower
Enter your guess (1-100): 60
Feedback: Higher

=== GAME RESULTS ===
Target number was: 63
Winner: player_1
```

2. Players guess out of bounds → get warning.

In this case, Player 2 entered a value outside the predefined range. So, no changes on the terminal of Player 1, Server and Player 2 terminals will change as follows:

Server Terminal

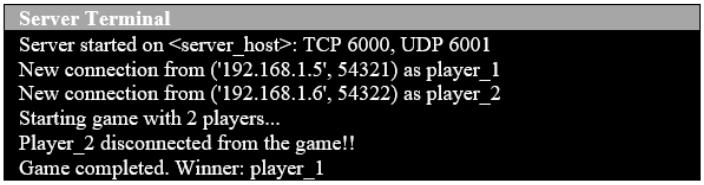
```
Server started on <server_host>: TCP 6000, UDP 6001
New connection from ('192.168.1.5', 54321) as player_1
New connection from ('192.168.1.6', 54322) as player_2
Starting game with 2 players...
Send Warning to player_2
Game completed. Winner: player_1
```

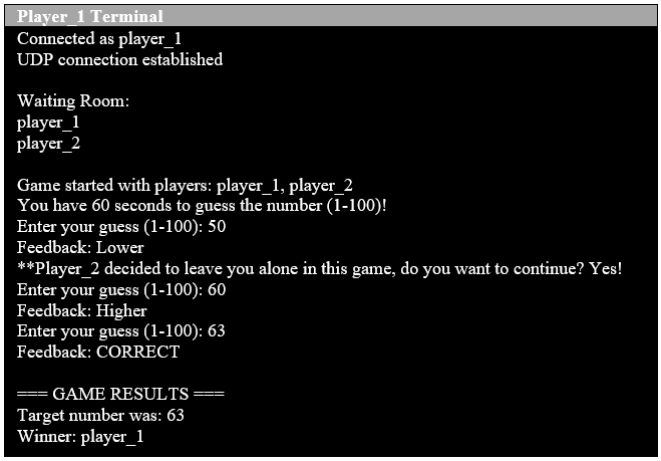
Player_2 Terminal

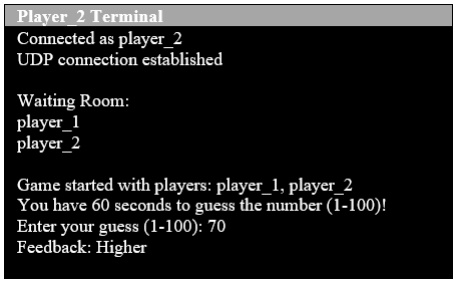
```
Connected as player_2
UDP connection established

Waiting Room:
player_1
player_2

Game started with players: player_1, player_2
You have 60 seconds to guess the number (1-100)!
Enter your guess (1-100): 110
Warning: Out of the range, miss your chance
Enter your guess (1-100): 50
Feedback: Lower
Enter your guess (1-100): 60
Feedback: Higher

=== GAME RESULTS ===
Target number was: 63
Winner: player_1
```

3. One player disconnects mid-game → game continues or ends.

Player 2 disconnected from the game, the terminals of the three as follows:







e) Initial Setting:

Here are the parameters of the game that you need to set before start and the permitted libraries that you can be imported (if needed) in your code:

- tcp_port = 6000
- udp_port = 6001
- server_name = based on your localhost (e.g., '0.0.0.0')
- min_players = 2
- max_players = 4
- to_enter_your_guess = 10 seconds
- game_duration = 60 seconds
- players_names:
  - (if group is more than one student) = {Player_1=lastName_firstName (for student1), Player_2=lastName_firstName (for student2), and so on}.
  - (if group is one student) = {Player_1=lastName_firstName (for student1), Player_2=firstName_ lastName (for student1)}.
- You are allowed to use these libraries = {socket, random, threading, time, collections}.

Organize the files (server.py and client.py) as outlined in ***Section B (Requirements and Deliverables)***.

D) Report and Grading Criteria

1) Project Report:

The report should include the following:

a) Cover page:
- Include the university logo, department, course name and number, project title, names, and IDs of all team members along with their section, and submission date.

b) Theory and Procedure:

- Explain the theoretical concepts relevant to each project requirement. Organize these ideas according to the specified titles for each section.
- List the components or methods used for each part of the project, including a brief explanation of why each one is necessary. Use diagrams or flowcharts to illustrate the solutions, with detailed descriptions and captions for all figures. Be sure to reference each figure in your descriptions.
- Cite sources for all theoretical concepts and ideas. Ensure all references are included at the end of the report.

c) Results and Discussions:
- For each task listed in ***Section C (Project Tasks)***, provide screenshots that show the outputs and results, along with a discussion. Screenshots of the code may be included to support your explanations, but avoid using screenshots alone without commentary.
- Discuss your method/solution and results in detail.
- For **Task 2**, provide browser screenshots that demonstrate the project's functionality for all required links. Test the task both from a browser on the same computer and from an external device (e.g., another computer or smartphone within your local network). Also, include a screenshot showing the HTTP request output printed on the command line.
- Ensure each screenshot includes **your system's date and time**.

d) Alternative Solutions, Issues, and Limitations:
- Discuss any alternative approaches that could potentially be used to solve the project tasks, beyond those specified in the instructions.
- Describe any limitations, issues, or challenges you encountered, either individually or as a team.
- Highlight any parts of the project that did not work as expected, if applicable.

e) Teamwork:
- Document the contributions of each team member using a chart that outlines the specific tasks completed by each member.

Ensure the report includes proper numbering, a table of contents, a list of figures, a list of tables, references, and appendices where needed.

2) Grading Criteria:

This project is worth **12%** of the total grade: **8%** will be allocated to the source code and report, while the remaining **4%** will be based on short-answer questions in the exam related to this project. Performance on this project will also be partially competitive, with additional points awarded for the following distinguishing factors:

a) Quality and completeness of functioning features.
b) The cohesiveness of the integrated components in achieving the project objectives.
c) Demonstrated understanding of implementation details.
d) User-friendliness and ease of interfacing with the components.
e) Deductions may apply if extensions or names deviate from those specified in ***Section B (Requirements and Deliverables)*** and ***Section C (Project Tasks)***.

Late submissions are permitted up to three days past the deadline, with a 10% deduction for each day late. **Helpful resources** to support your success on this project are available at the following links:

[1] https://drive.google.com/drive/folders/1ZTSFxnkihq746bTk9uZVzYVVgxXcIMD?usp=drive_link
[2] https://www.naukri.com/code360/library/socket-programming-with-multithreading-in-python
[3] https://eecs485staff.github.io/p4-mapreduce/threads-sockets.html
[4] https://www.youtube.com/watch?v=IEEhzQoKtQU
[5] https://www.youtube.com/watch?v=xceTFWy_eag