Pokhara University
Faculty of Science and Technology

Course Code.: CMP 380	Full marks: 100
Course title: **Network Programming (3-0-2)**	Pass marks: 45
Nature of the course: Theory/Practice/Theory & Practice	Time per period: 1 hour
Year, Semester:III,I	Total periods: 45
Level: Bachelor	Program: BE

1. Course Description

Computer network programming involves writing computer programs that enable process to communicate with each other across a computer network or within same system. Network programming is client-server or peer-to-peer programming so to make two processes to communicate with each other one process must take the initiative while the other is waiting for it.

This course provides an in-depth understanding of network programming, focusing on the principles and practices required to build robust and scalable network applications. Students will explore both Unix and Windows network programming environments, with a strong emphasis on socket programming, I/O models, multithreading, and network protocols. The course begins with fundamental concepts of network programming and progresses to advanced topics such as signal handling, daemon processes, and secure communication. Additionally, students will gain exposure to the latest trends and emerging technologies in network programming, including real-time communication protocols and software-defined networking. The course also includes practical lab exercises that reinforce the theoretical knowledge and develop hands-on skills in implementing networked applications.

2. Course Objectives

The general objectives of this course includes

- In-depth understanding of network programming principles and practices, developing client-server process communication applications in both unix and windows environment.

The course is designed with the following specific objectives:

- To grasp the basic concepts and importance of network programming in both Unix and Windows environments.
- To develop proficiency in Unix socket programming, including the creation and management of TCP and UDP connections.
- To understand and implement various I/O models such as blocking, non-blocking, and asynchronous I/O.
- To gain the ability to handle signals and develop daemon processes in Unix.
- To utilize Unix logging facilities like Syslog for monitoring and debugging network applications.
- To acquire the skills to create and manage network connections using the Winsock API in a Windows environment.

- To understand the concepts and techniques required for developing multithreaded and concurrent programming.
- To Learn about the latest protocols and technologies such as WebSockets and gRPC networking protocols.
- To engage in hands-on lab exercises that reinforce theoretical concepts and develop practical skills in Unix and Winsock network programming.
- To build real-world networked applications (standalone and cross platform) and troubleshoot common network issues.
- To design and implement the client-server or P2P based system which is able to communicate across the different network platforms without depending on operating system architecture.

3. Contents in Detail

Specific Objectives	**Contents**
• To understand and review the basic concepts of computer network and network Programming. • To understanding basic working mechanisms of client server model in process level. • To review TCP/IP model and compare it with ISO OSI model. • To understand the characteristics, purposes, and appropriate use cases of basic network protocols such as TCP, IP, UDP, and SCTP. • To analyze the TCP state transition diagram to understand the various states in a TCP connection,termination and how data transmission is managed. • To compare basic protocols such TCP, UDP, and SCTP • To understand the basic concept of sockets and their types.	**Unit I: Network Programming Fundamentals (5 hours)** 1.1 Introduction and importance of networking and network programming. 1.2 Fundamentals of Client server and P2P models 1.3 Basic Network Protocols 1.3.1 Characteristics and use cases of TCP, IP, UDP, SCTP 1.4 TCP state transition diagram. 1.5 Comparisons of protocols (TCP,UDP,SCTP) 1.6 Introduction to Sockets 1.6.1 Concept of sockets in network communication 1.6.2 Types of sockets: Stream (TCP) vs Datagram (UDP)
• To understand and explain the concept and importance of Unix sockets in network communication, including how they provide an interface for inter-process communication (IPC) within Unix systems. • To understand and utilize various socket address structures in network programming • To compare and contrast these	**Unit II: Basics of Unix Network Programming (12 hours)** 2.1 Overview of Unix OS and Network Administration 2.2 Unix socket introduction 2.3 Socket Address Structures 2.3.1 Genric socket: struct sockaddr 2.3.2 IPv4: struct sockaddr_in 2.3.3 IPv6: struct sockaddr_in6

structures, understanding their specific use cases and applications. • To apply the concept of value-result arguments in socket programming, particularly how they affect the behavior and output of system calls. • To understand and implement byte ordering functions to ensure proper data transmission across different systems. • To understand the concepts of hostname and network name lookups using system functions to resolve human-readable addresses into network addresses, and understand the role of DNS in this process. • To use elementary TCP and UDP socket system calls to create basic network communication between clients and servers. • To implement Unix domain sockets for local inter-process communication (IPC), including the ability to pass file descriptors between processes. • To understand the concept of signals in Unix, including their use in managing process behavior • To implement signal handling in network applications to respond to asynchronous events, such as interrupts (SIGINT) and I/O signals (SIGIO) • To Identify the characteristics of daemon processes in Unix and understand their role in long-running background tasks.	2.3.4 Unix Domain Sockets: struct sockaddr_un 2.3.5 Storage: struct sockaddr_storage 2.4 Comparisions of various socket address structure 2.5 Value result arguments 2.6 Byte ordering and manipulation functions 2.7 Hostname and Network Name Lookups 2.4 Basic socket system calls 2.4.1 Elementary TCP Sockets 2.4.2 Elementary UDP Sockets 2.5 Unix Domain Socket 2.5.1 Passing file descriptors 2.6 Signal Handling in Unix 2.6.1 Introduction to signals 2.6.2 Handling signals in network applications (e.g., SIGINT, SIGIO) 2.6.3 Signal functions (signal(), sigaction()) 2.7 Daemon Processes 2.7.1 Characteristics of daemon processes 2.7.2 Converting a program into a daemon 2.7.3 Daemonizing techniques in Unix
• To understand and Implement Various I/O Models in Unix: • To Implement concurrent server models using fork() for multi-client handling and select() for managing multiple socket descriptors. • To develop multi-threaded network applications using the pthreads library to improve server performance and responsiveness.	**Unit III: Advance Unix Network Programming (12 hours)** 3.1 I/O Models in Unix 3.1.1 Blocking I/O 3.1.2 Non-blocking I/O 3.1.3 I/O multiplexing (select(), pselect() poll()) 3.1.4 Signal-driven I/O 3.1.5 Asynchronous I/O

• To apply socket programming techniques to send and receive broadcast and multicast messages across a network. • To understand and apply socket options using setsockopt(), getsockopt(), fcntl(), and ioctl() to modify and control socket behavior. • To Understand the role and importance of Syslog in Unix systems for logging network application events and messages. • To define and understand and implement the core concepts and challenges of network security in Unix network programming.	3.2 Concurrent Server Design 3.2.1 Overview of process and threads 3.2.2 Fork() and exec() function 3.2.3 Using fork() to handle multiple clients 3.2.4 Using select() to handle multiple socket descriptors 3.2.5 Multithreading model using pthreads 3.3 Implementing broadcast and multicast communication 3.4 Socket Options 3.4.1 Using setsockopt(), getsockopt(), fcntl() and ioctl() to modify socket behavior 3.4.2 Common options: SO_REUSEADDR, SO_BROADCAST, SO_KEEPALIVE, SO_LINGER etc. 3.5 Logging in Unix 3.5.1 Introduction to Syslog 3.5.2 Logging messages from network applications 3.5.3 Configuring and using syslog(), openlog(), and closelog() 3.6 Socket operations 3.7 Introduciton to P2P programming 3.8 P2P Socket fundamentals 3.9 Overview Network Security Programming 3.9.1 Defining Security 3.9.2 Challenges of Security 3.9.3 Securing by Hostname or Domain Name 3.9.4 Identification by IP Number 3.9.5 Wrapper program to implement simple security policy
• To gain a foundational understanding of the Winsock API, including its role and importance in network programming on Windows platforms. • Identify and articulate the key differences between Unix and Winsock programming • To develop basic server applications in the Winsock environment. • To Implement and manage blocked I/O operations • To utilize asynchronous database and I/O functions within Winsock, enabling non-blocking communication and efficient data handling in network	**Unit IV: Basics of Winsock Network Programming (6 hours)** 4.1. Introduction to Winsock 4.1.1 Overview of the Winsock API 4.1.2 Differences between Unix and Winsock programming 4.1.3 Winosock DLL 4.1.3.1 Setting Up Winsock Environment 4.1.3.2 Initialization: WSAStartup() 4.1.3.3 Clean-up: WSACleanup() 4.2 Basic Winsock API Functions 4.2.1 Socket creation and binding (socket(), bind()) 4.2.2 Listening and accepting connections

applications. • To create simple TCP and UDP client-server models in the Winsock environment	(listen(), accept()) 4.2.3 Sending and receiving data (send(), recv()) 4.2.4 Closing a socket (closesocket()) 4.2.5 Functions for handling Blocked IO 4.2.6 Asynchronous IO functions 4.2.7 Creating Simple TCP/UDP Clients and Servers programs using Winsock
• To understand the error handling functions specific to asynchronous and nonblocking operations in Winsock. • To utilize nonblocking sockets to create responsive network applications that can handle multiple operations without getting blocked. • To grasp the basics of overlapped I/O and implement it in network applications to achieve nonblocking and asynchronous data transfers. • To understand the concepts of advanced polling mechanisms using WSAPoll() • To manage threads within Winsock applications, understanding the creation, synchronization, and communication between threads in a networked environment. • To develop and implement basic cross-platform network applications that can run on both Unix and Windows environments	**Unit V: Advanced Winsock Programming (5 hours)** 5.1 Asynchronous I/O and Nonblocking operations in Winsock 5.1.1 Error handling functions 5.1.2 Using non blocking sockets 5.1.3 Select in conjuction with accept, select with recv/recvfrom and send/sendto 5.1.4 Overlapped I/O: Basics and implementation 5.1.5 Event-driven programming 5.1.5.1 Using WSAEventSelect() 5.1.5.2 Completion routines 5.2 Winsock Extensions 5.2.1 Advanced polling mechanisms: WSAPoll() 5.2.2 Event management with WSAEventSelect() 5.3 Implementation of basic cross platform application
• To Understand and use common network utilities • To explain and implement the concept of WebSockets and their role in full-duplex communication • To understand the gRPC framework and and develop gRPC-based applications • To gain an introduction to secure socket programming using TLS (Transport Layer Security) and SSL (Secure Sockets Layer). • To understand and apply fundamental	**Unit VI: Network utilities, Current Trends and Emerging Technologies in Network Programming (5 hours)** 6.1 Network Utilities and Applications 6.1.1 Introduction to ping, telnet, ip/ifconfig, iperf, netstat, remote login 6.2 Real-Time Communication Protocols 6.2.1 WebSockets: Full-duplex communication over a single TCP connection 6.2.2 gRPC: High-performance RPC framework 6.3 Security in Network Programming 6.3.1 TLS/SSL: Introduction to secure socket programming

cryptography concepts, including encryption, hashing, and certificates. • To understand the core concepts and principles of Software-Defined Networking, including network programmability and centralized control.	6.4 Software-Defined Networking (SDN) 6.4.1 Overview of SDN concepts 6.4.2 Introduction to OpenFlow protocol and SDN controllers 6.4.3 Introduction to P4 and frenetic programming

4. Methods of Instruction

The Network Programming course will employ a comprehensive approach to instruction, integrating theoretical knowledge with practical application. Interactive lectures will introduce core concepts of Unix and Winsock programming, while hands-on lab session, group based project work and demonstrations will provide real-time/real life examples of these principles in action. Hands-on lab sessions will reinforce learning, allowing students to implement network applications and security protocols. Real-world case studies and group projects will foster collaboration and critical thinking.This multifaceted approach aims to equip students with both the theoretical foundation and practical skills necessary for proficient network programming in a real-world context.

5. List of Practical Works

Below are the few listed practical work(Lab session) but not limited too. Practical work should be implemented using C/C++ programming language. It is not compulsory to implement websocket and gRPC using C, an instructor/student can choose any current trends language (node.js, python etc) for this.

1. Implementation of basic TCP Client-Server application in Unix.
2. Implementation of basic UDP Client-Server Application in Unix.
3. Implementation of concurrent Server with `fork()` in Unix
4. Implementation of handling multiple descriptors using select() system call.
5. Implementation multithreaded Server in Unix
6. Implementations of various IO models including blocking, non-blocking, multiplexing and signal driven IO.
7. Implementation of Signal Handling in a Network Program
8. Implementation of Daemonizing a process.
9. Implementation of basic network security programs.
10. Basic Winsock TCP Client-Server Application.
11. Implementation of Secure Socket Programming.
12. Implementation of basic real time communication protocols like websocket, gRPC using node.js, python.

6. Evaluation system and Students' Responsibilities

Evaluation System

In addition to the formal exam(s) conducted by the Office of the Controller of Examination of Pokhara University, the internal evaluation of a student may consist of class attendance, class participation, quizzes, assignments, presentations, written exams, etc.

In this course, students will be required to develop a network application as part of their subject project. This project will involve designing, implementing, and testing a network application that can operate across in Unix/Windows or both platform.. The project will require students to apply the concepts and techniques learned throughout the course.By completing this project, students will gain hands-on experience in building real-world network applications, reinforcing their understanding of course material and preparing them for professional challenges in network programming.

The tabular presentation of the evaluation system is as follows.

Internal Evaluation	**Weight**	**Marks**	**External Evaluation**	**Marks**
Theory		30	Semester-End examination	50
Attendance & Class Participation	10%			
Assignments	20%			
Presentations/Quizzes	10%			
Internal Assessment	60%			
Practical		20		
Attendance & Class Participation	10%			
Lab Report/Project Report	20%			
Practical Exam	20%			
Project Work	40%			
Viva	10%			
Total Internal		50		
Full Marks: 50 + 50 = 100				

Students' Responsibilities:
Each student must secure at least 45% marks in the internal evaluation with 80% attendance in the class to appear in the Semester End Examination. Failing to obtain such score will be given NOT QUALIFIED (NQ) and the student will not be eligible to appear in the End-Term examinations. Students are advised to attend all the classes and complete all the assignments within the specified time period. If a student does not attend the class(es), it is his/her sole responsibility to cover the topic(s) taught during the period. If a student fails to attend a formal exam, quiz, test, etc. there won't be any provision for a re-exam.

Prescribed Books and References

- “Unix Network Programming, Volume 1: The Sockets Networking API" by W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff
- "Advanced Programming in the UNIX Environment" by W. Richard Stevens, Stephen A. Rago:
- “Linux Socket Programming” by Example by Warren W. Gay
- "TCP/IP Sockets in C: Practical Guide for Programmers" by Michael J. Donahoo and Kenneth L. Calvert
- "Network Programming for Microsoft Windows" by Anthony Jones and Jim Ohlund

Notes/Online Resources:

- Beej's Guide to Network Programming:
- "The Complete Windows Socket Programming Tutorial" by Bob Quinn and Dave Shute:
- Official gRPC Documentation:
- The Complete Windows Socket Programming Tutorial" by Bob Quinn and Dave Shute