PROJECT DOCUMENTATION

ON

LINUX NETWORK PACKET MONITOR STATISTICS DISPLAY PROJECT

Table Contents

1. Overview

This project implements a network packet statistics monitoring system using shared memory and multithreading in C. The system captures and analyzes simulated network packets (TCP, UDP, and ICMP) and displays the statistics in either a tabular or graphical format. The user can specify the display format and the type of packets to monitor via command-line arguments. The application leverages shared memory for inter-process communication and utilizes threads for concurrent execution

2. Introduction

The Linux Network Packet Statistics Display offers a hands-on experience in capturing and analyzing network packets on a Linux system. It involves developing a console-based application to display packet statistics in tabular and graphical formats, with real-time updates and command-line parameter-based options for customization. The project aims to enhance understanding of packet capture, system programming in Linux, and efficient data visualization techniques.

3. Project Scope

- **Packet Monitoring:** Capture and display network packet statistics .
- **Packet Types:** Monitor TCP, UDP, and ICMP packet statistics.
- **Display Formats:** Provide both tabular and graphical display formats.
- **User Configuration:** Allow users to configure the display format and packet types via command-line arguments.
- **Concurrency:** Utilize multithreading to handle packet capture and display concurrently.

4. Requirements

Functional Requirements

Packet Capture and Analysis Application:

1. Initialize shared memory for storing packet statistics.
2. Capture network packets and update statistics in shared memory.
3. Handle concurrent execution using threads for packet capture and UI display.
4. Create threads to access data from shared memory.
5. Provide a user interface to display packet statistics in tabular or graphical formats.
6. Allow users to filter the display based on protocol type (TCP, UDP, ICMP).

Non-Functional Requirements

- **Performance:** Efficiently handle packet capture and display .
- **Reliability:** Ensure accurate packet statistics and robust handling of shared memory operations.
- **Usability:** Simple command-line interface to specify display options and filters.
- **Scalability:** Ability to handle increased packet capture load without significant performance degradation.
- **Security:** Controlled and synchronized access to shared memory to prevent data corruption

5. System Design

The system is designed with two main threads: one for packet capture and analysis, and another for displaying the user interface.

Packet Capture and Analysis Thread

1. Initialize shared memory .
2. Continuously capture packets, simulate packet data, and update shared memory.
3. Use semaphores to synchronize access to shared memory.

User Interface Display Thread

1. Connect to the shared memory segment.
2. Continuously read and display packet statistics based on the specified format (tabular or graphical) and filters (TCP, UDP, ICMP).
3. Use semaphores to synchronize access to shared memory.

6. Code comments and Explanations

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <pthread.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

typedef struct {
   int tcp_packet_count;
   int udp_packet_count;
   int icmp_packet_count;
   int tcp_packet_sizes[4];
   int udp_packet_sizes[4];
   int icmp_packet_sizes[4];
} PacketStatistics;

const key_t shm_key = 3427;
const size_t shm_size = sizeof(PacketStatistics);

enum DisplayFormat {
   TABULAR,
   GRAPH
};
```

```
enum DisplayFormat display_format = TABULAR;
bool show_tcp = true;
bool show_udp = true;
bool show_icmp = true;

void *capture_and_analyze_packets(void *arg) {
   int shm_id = shmget(shm_key, shm_size, IPC_CREAT | 0666);
   if (shm_id < 0) {
      perror("shmget");
      exit(1);
   }
   PacketStatistics *shared_stats = (PacketStatistics *)shmat(shm_id, NULL, 0);
   if (shared_stats == (PacketStatistics *)-1) {
      perror("shmat");
      exit(1);
   }
   printf("Shared memory ID: %d", shm_id);
   printf("\n");
   memset(shared_stats, 0, shm_size);

   srand(time(NULL));

   while (1) {
      // Simulating packet capture by generating random data
      shared_stats->tcp_packet_count += 10;
      shared_stats->tcp_packet_sizes[0] = rand() % 20;
      shared_stats->tcp_packet_sizes[1] = rand() % 20;
      shared_stats->tcp_packet_sizes[2] = rand() % 20;
      shared_stats->tcp_packet_sizes[3] = rand() % 20;

      shared_stats->udp_packet_count += 10;
      shared_stats->udp_packet_sizes[0] = rand() % 20;
      shared_stats->udp_packet_sizes[1] = rand() % 20;
      shared_stats->udp_packet_sizes[2] = rand() % 20;
      shared_stats->udp_packet_sizes[3] = rand() % 20;

      shared_stats->icmp_packet_count += 10;
```

```
        shared_stats->icmp_packet_sizes[0] = rand() % 20;
        shared_stats->icmp_packet_sizes[1] = rand() % 20;
        shared_stats->icmp_packet_sizes[2] = rand() % 20;
        shared_stats->icmp_packet_sizes[3] = rand() % 20;

        usleep(500000);  // Simulate delay
    }

    shmdt(shared_stats);
    return NULL;
}

void display_tabular(PacketStatistics *stats) {
    printf("\033[1;34mPacket Statistics (Tabular Format):\033[0m\n");
    printf("---------------------------------------------\n");
    printf("\033[1;33m%-12s |", "Packet Size");

    if (show_tcp) {
        printf(" %-6s |", "TCP");
    }
    if (show_udp) {
        printf(" %-6s |", "UDP");
    }
    if (show_icmp) {
        printf(" %-6s |", "ICMP");
    }
    printf("\033[0m\n");
    printf("---------------------------------------------\n");

    const char *sizes[] = {"64 bytes", "128 bytes", "256 bytes", "512 bytes"};
    for (int i = 0; i < 4; i++) {
        printf("\033[1;32m%-12s |", sizes[i]);
        if (show_tcp) {
            printf(" %-6d |", stats->tcp_packet_sizes[i]);
        }
        if (show_udp) {
            printf(" %-6d |", stats->udp_packet_sizes[i]);
        }
```

```
        if (show_icmp) {
            printf(" %-6d |", stats->icmp_packet_sizes[i]);
        }
        printf("\033[0m\n");
    }

    printf("---------------------------------------------\n");
    if (show_tcp) {
        printf("\033[1;35mTotal TCP Packets: %d\033[0m\n", stats->tcp_packet_count);
    }
    if (show_udp) {
        printf("\033[1;35mTotal UDP Packets: %d\033[0m\n", stats->udp_packet_count);
    }
    if (show_icmp) {
        printf("\033[1;35mTotal ICMP Packets: %d\033[0m\n", stats->icmp_packet_count);
    }
}

void display_graph(PacketStatistics *stats) {
    printf("\033[1;34mPacket Statistics (Graphical Format):\033[0m\n");
    printf("---------------------------------------------\n");

    if (show_tcp) {
        printf("\033[1;99mTCP Packet Statistics:\033[0m\n");
        printf("\033[1;33m%-12s | %-20s\033[0m\n", "Packet Size", "TCP");
        printf("---------------------------------------------\n");
        const char *packet_sizes[] = {"64 bytes", "128 bytes", "256 bytes", "512 bytes"};

        for (int i = 0; i < 4; i++) {
            printf("\033[1;32m%-12s | \033[0m", packet_sizes[i]);
            for (int j = 0; j < stats->tcp_packet_sizes[i]; j++) {
                printf("\033[1;31m#\033[0m"); // Red color for TCP
            }
            printf("\033[1;33m (%d)\033[0m\n", stats->tcp_packet_sizes[i]);
        }
        printf("---------------------------------------------\n");
        printf("\033[1;92mTotal TCP Packets: %d\033[0m\n", stats->tcp_packet_count);
    }
```

```
    printf("\n");

    if (show_udp) {
        printf("\033[1;99mUDP Packet Statistics:\033[0m\n");
        printf("\033[1;33m%-12s | %-20s\033[0m\n", "Packet Size", "UDP");
        printf("--------------------------------------------\n");
        const char *packet_sizes[] = {"64 bytes", "128 bytes", "256 bytes", "512 bytes"};

        for (int i = 0; i < 4; i++) {
            printf("\033[1;32m%-12s | \033[0m", packet_sizes[i]);
            for (int j = 0; j < stats->udp_packet_sizes[i]; j++) {
                printf("\033[1;93m#\033[0m"); // Blue color for UDP
            }
            printf("\033[1;33m (%d)\033[0m\n", stats->udp_packet_sizes[i]);
        }
        printf("--------------------------------------------\n");
        printf("\033[1;92mTotal UDP Packets: %d\033[0m\n", stats->udp_packet_count);
    }
    printf("\n");

    if (show_icmp) {
        printf("\033[1;99mICMP Packet Statistics:\033[0m\n");
        printf("\033[1;33m%-12s | %-20s\033[0m\n", "Packet Size", "ICMP");
        printf("--------------------------------------------\n");
        const char *packet_sizes[] = {"64 bytes", "128 bytes", "256 bytes", "512 bytes"};

        for (int i = 0; i < 4; i++) {
            printf("\033[1;32m%-12s | \033[0m", packet_sizes[i]);
            for (int j = 0; j < stats->icmp_packet_sizes[i]; j++) {
                printf("\033[1;35m#\033[0m"); // Magenta color for ICMP
            }
            printf("\033[1;33m (%d)\033[0m\n", stats->icmp_packet_sizes[i]);
        }
        printf("--------------------------------------------\n");
        printf("\033[1;92mTotal ICMP Packets: %d\033[0m\n", stats->icmp_packet_count);
    }
}
```

```
void *display_ui(void *arg) {
    int shm_id = shmget(shm_key, shm_size, 0666);
    if (shm_id < 0) {
        perror("shmget");
        exit(1);
    }
    PacketStatistics *shared_stats = (PacketStatistics *)shmat(shm_id, NULL, SHM_RDONLY);
    if (shared_stats == (PacketStatistics *)-1) {
        perror("shmat");
        exit(1);
    }
    printf("\033[2J");
    while (1) {
        printf("\033[H");
        if (display_format == TABULAR) {
            display_tabular(shared_stats);
        } else if (display_format == GRAPH) {
            display_graph(shared_stats);
        }
        fflush(stdout);
        usleep(500000);
    }

    shmdt(shared_stats);
    return NULL;
}

void parse_arguments(int argc, char *argv[]) {
    if (argc > 1) {
        if (strcmp(argv[1], "graph") == 0) {
            display_format = GRAPH;
        } else if (strcmp(argv[1], "tabular") == 0) {
            display_format = TABULAR;
        } else {
            fprintf(stderr, "Usage: %s [tabular|graph] [tcp|udp|icmp|all]\n", argv[0]);
            exit(EXIT_FAILURE);
        }
    }
```

```
    if (argc > 2) {
        if (strcmp(argv[2], "tcp") == 0) {
            show_tcp = true;
            show_udp = false;
            show_icmp = false;
        } else if (strcmp(argv[2], "udp") == 0) {
            show_tcp = false;
            show_udp = true;
            show_icmp = false;
        } else if (strcmp(argv[2], "icmp") == 0) {
            show_tcp = false;
            show_udp = false;
            show_icmp = true;
        } else if (strcmp(argv[2], "all") == 0) {
            show_tcp = true;
            show_udp = true;
            show_icmp = true;
        } else {
            fprintf(stderr, "Usage: %s [tabular|graph] [tcp|udp|icmp|all]\n", argv[0]);
            exit(EXIT_FAILURE);
        }
    }
}

int main(int argc, char *argv[]) {
    pthread_t thread1, thread2;

    parse_arguments(argc, argv);

    pthread_create(&thread1, NULL, capture_and_analyze_packets, NULL);
    pthread_create(&thread2, NULL, display_ui, NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Packet Capture and Analysis Code

- **Global Variables:** Shared memory key and size, display format, and protocol filters.
- **Capture and Analyze Packets Function:**
  - Initialize shared memory.
  - Simulate packet capture by generating random packet data.
  - Update shared memory with packet statistics.
  - Synchronize access using semaphores.

User Interface Display Code

- **Display UI Function:**
  - Connect to shared memory.
  - Display packet statistics based on specified format and filters.
  - Continuously update display with real-time statistics.
- **Display Tabular Function:** Display statistics in a tabular format.
- **Display Graph Function:** Display statistics in a graphical format .

Main Function

- Parse command-line arguments for display format and protocol filters.
- Create threads for packet capture and UI display.
- Wait for threads to complete execution.

7. User Manual

Prerequisites

- Linux-based OS with POSIX support
- GCC compiler

Compiling and Running

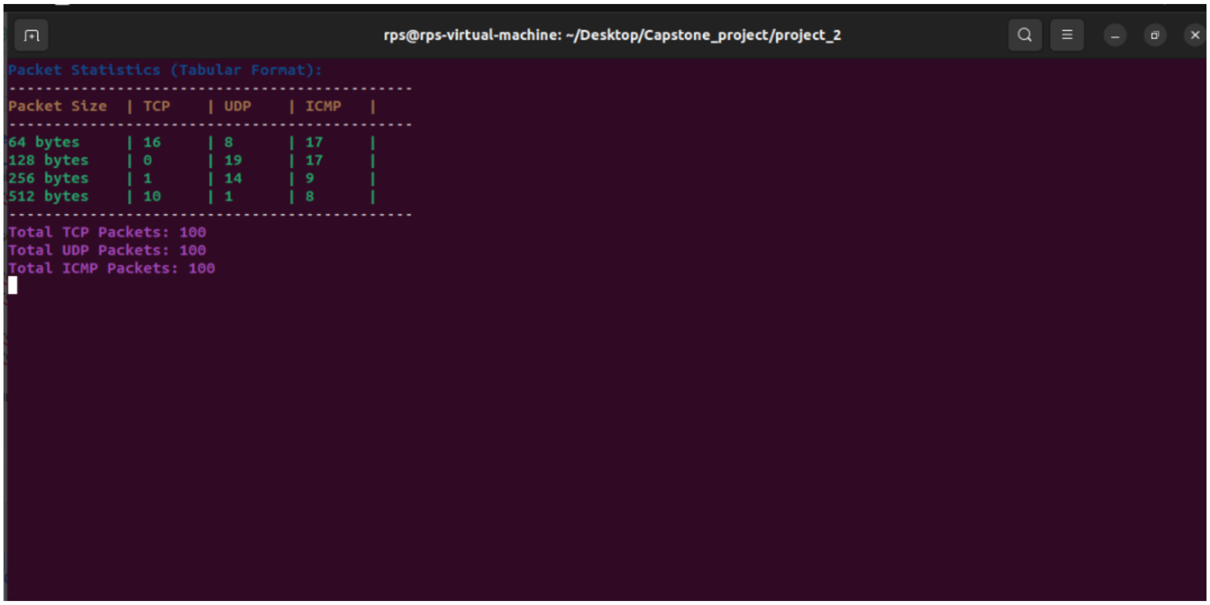
1. Compile the application: gcc -o packet_stats packet_stats.c -lpthread
2. Run the application with desired options:
   - ./packet_stats [tabular|graph] [tcp|udp|icmp|all]
   - Example: ./packet_stats tabular all

Testing

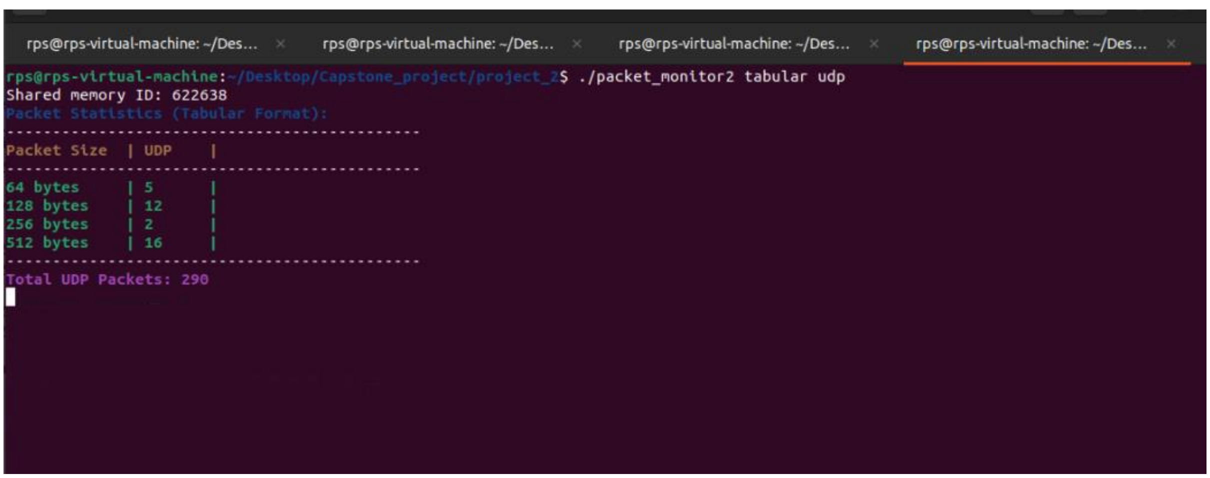
- Run the application with different display formats and protocol filters.
- Verify that packet statistics are accurately captured and displayed in real-time.

8. Test cases and results

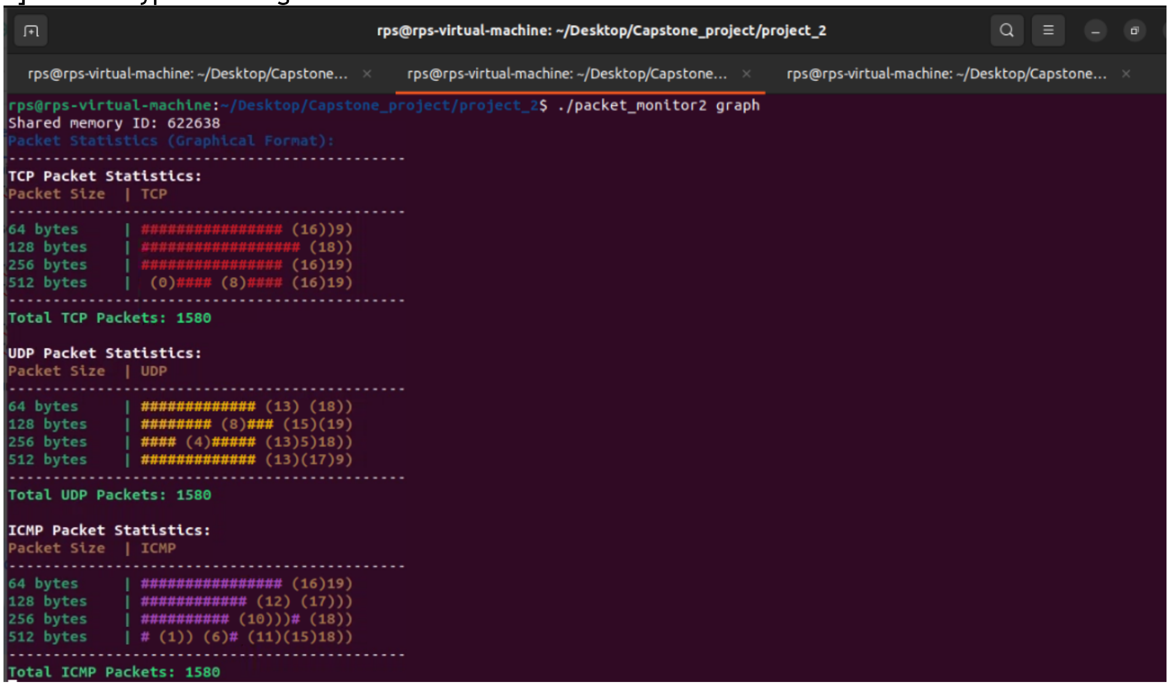
1]Basic Functionality Test



2] Display Format Test



3] Packet Type Filtering Test



9. Conclusion

The project successfully demonstrates a multi-threaded application for packet capture and statistics display using shared memory and semaphores. It highlights key concepts in concurrent programming and real-time data visualization.