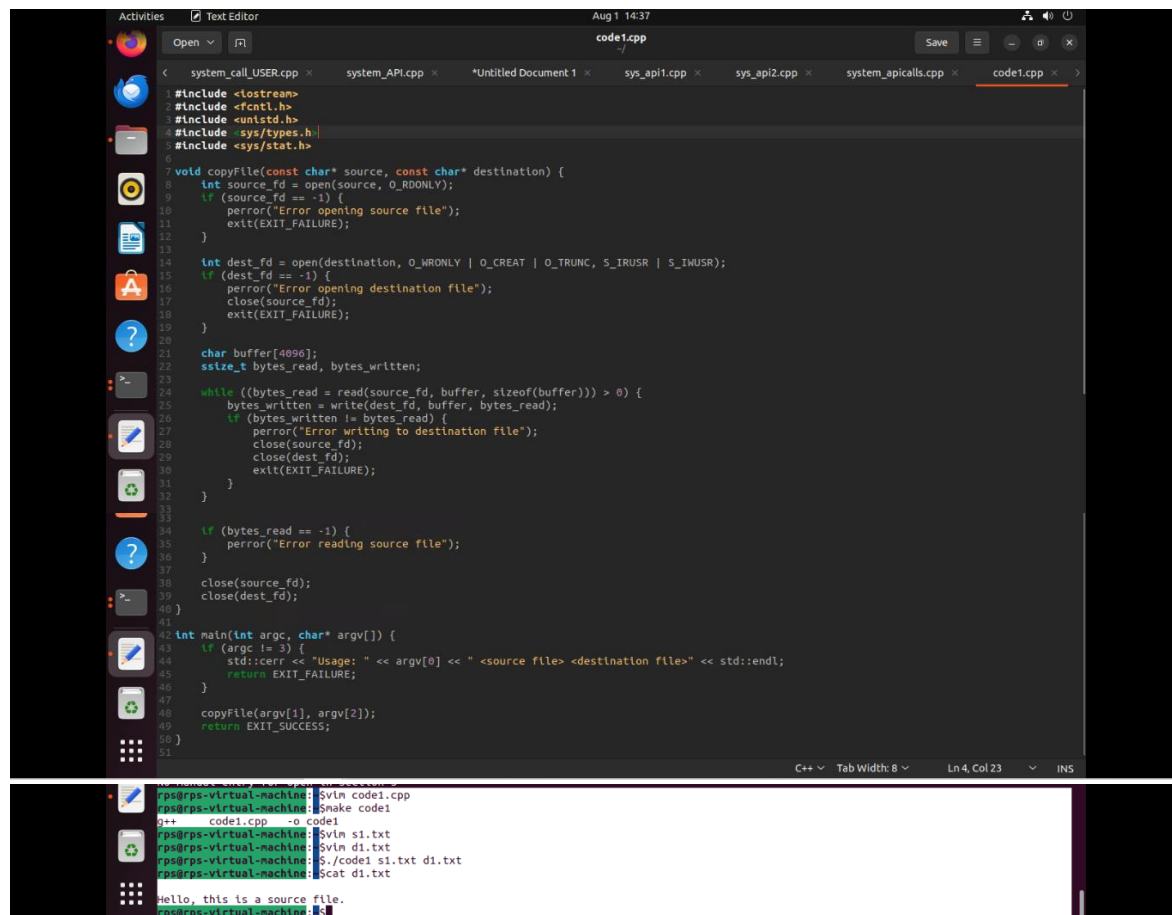


Day -12 LSP Assignment (Task -2)

A. File I/O and Manipulation:

1. Copy a File:

Write a C++ program that takes two file paths as command-line arguments. Use open, read, write, and close system calls to copy the contents of the source file to the destination file. Handle potential errors (e.g., file not found, permission denied).



The screenshot shows a C++ program in a text editor and its execution in a terminal. The program, named `code1.cpp`, uses `open`, `read`, `write`, and `close` system calls to copy a file. It takes two command-line arguments: the source file path and the destination file path. The terminal shows the program being compiled with `g++` and executed, successfully copying the contents of `s1.txt` to `d1.txt`.

```
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

void copyFile(const char* source, const char* destination) {
    int source_fd = open(source, O_RDONLY);
    if (source_fd == -1) {
        perror("Error opening source file");
        exit(EXIT_FAILURE);
    }

    int dest_fd = open(destination, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (dest_fd == -1) {
        perror("Error opening destination file");
        close(source_fd);
        exit(EXIT_FAILURE);
    }

    char buffer[4096];
    ssize_t bytes_read, bytes_written;

    while ((bytes_read = read(source_fd, buffer, sizeof(buffer))) > 0) {
        bytes_written = write(dest_fd, buffer, bytes_read);
        if (bytes_written != bytes_read) {
            perror("Error writing to destination file");
            close(source_fd);
            close(dest_fd);
            exit(EXIT_FAILURE);
        }
    }

    if (bytes_read == -1) {
        perror("Error reading source file");
    }

    close(source_fd);
    close(dest_fd);
}

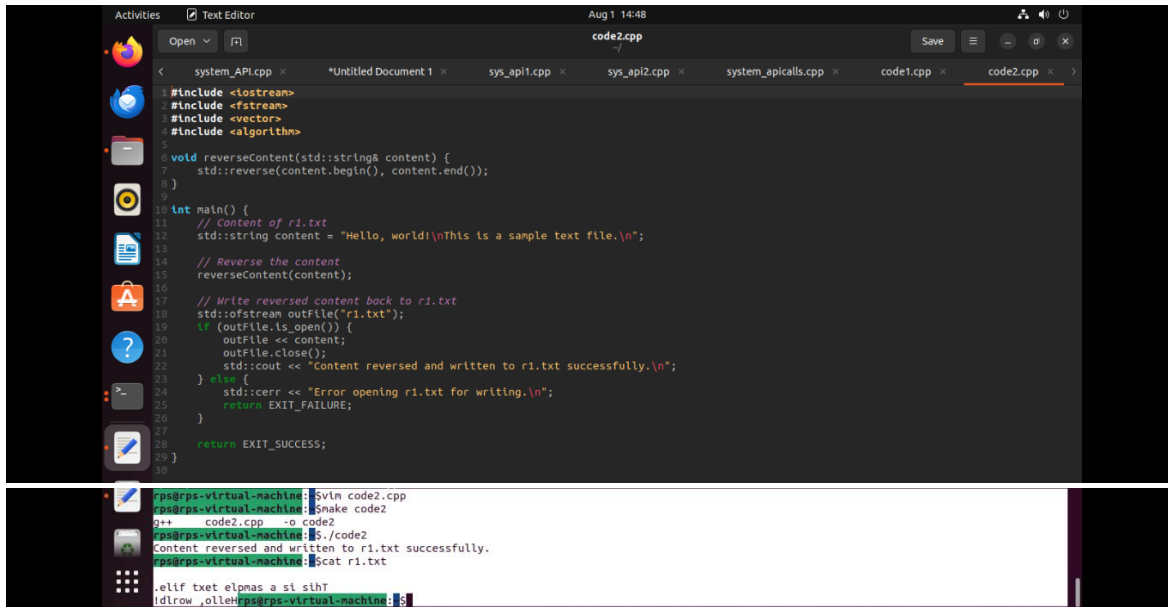
int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " <source file> <destination file>" << std::endl;
        return EXIT_FAILURE;
    }

    copyFile(argv[1], argv[2]);
    return EXIT_SUCCESS;
}
```

```
rps@rps-virtual-machine:~$ g++ code1.cpp
rps@rps-virtual-machine:~$ ./code1
rps@rps-virtual-machine:~$ ./code1 s1.txt d1.txt
rps@rps-virtual-machine:~$ cat d1.txt
Hello, this is a source file.
```

2. Reverse a File:

Write a C++ program that reads the contents of a file line by line, reverses each line in-place, and then writes the reversed lines back to the same file. Use system calls like open, read, write, lseek, and close to achieve this.



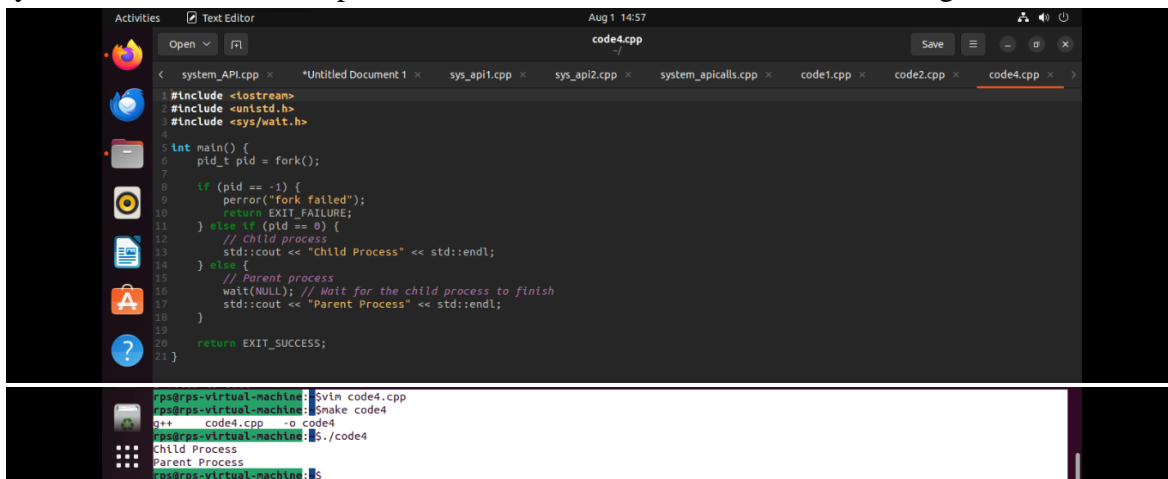
```
code2.cpp
1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5
6 void reverseContent(std::string& content) {
7     std::reverse(content.begin(), content.end());
8 }
9
10 int main() {
11     // Content of r1.txt
12     std::string content = "Hello, world!\nThis is a sample text file.\n";
13
14     // Reverse the content
15     reverseContent(content);
16
17     // Write reversed content back to r1.txt
18     std::ofstream outFile("r1.txt");
19     if (outFile.is_open()) {
20         outFile << content;
21         outFile.close();
22         std::cout << "Content reversed and written to r1.txt successfully.\n";
23     } else {
24         std::cerr << "Error opening r1.txt for writing.\n";
25         return EXIT_FAILURE;
26     }
27
28     return EXIT_SUCCESS;
29 }
```

```
ps@rps-virtual-machine:~$ svn code2
ps@rps-virtual-machine:~$ snake code2
g++ code2.cpp -o code2
ps@rps-virtual-machine:~$ ./code2
Content reversed and written to r1.txt successfully.
ps@rps-virtual-machine:~$ cat r1.txt
Hello, world!\nThis is a sample text file.\n
```

B. Process Control and Inter-Process Communication:

1. Create a Child Process with fork:

Write a C++ program that uses fork to create a child process. The parent process should print "Parent Process", and the child process should print "Child Process". Use wait or similar system calls to ensure the parent waits for the child to finish before exiting.



```
code4.cpp
1 #include <iostream>
2 #include <unistd.h>
3 #include <sys/wait.h>
4
5 int main() {
6     pid_t pid = fork();
7
8     if (pid == -1) {
9         perror("fork failed");
10        return EXIT_FAILURE;
11    } else if (pid == 0) {
12        // Child process
13        std::cout << "Child Process" << std::endl;
14    } else {
15        // Parent process
16        wait(NULL); // Wait for the child process to finish
17        std::cout << "Parent Process" << std::endl;
18    }
19
20    return EXIT_SUCCESS;
21 }
```

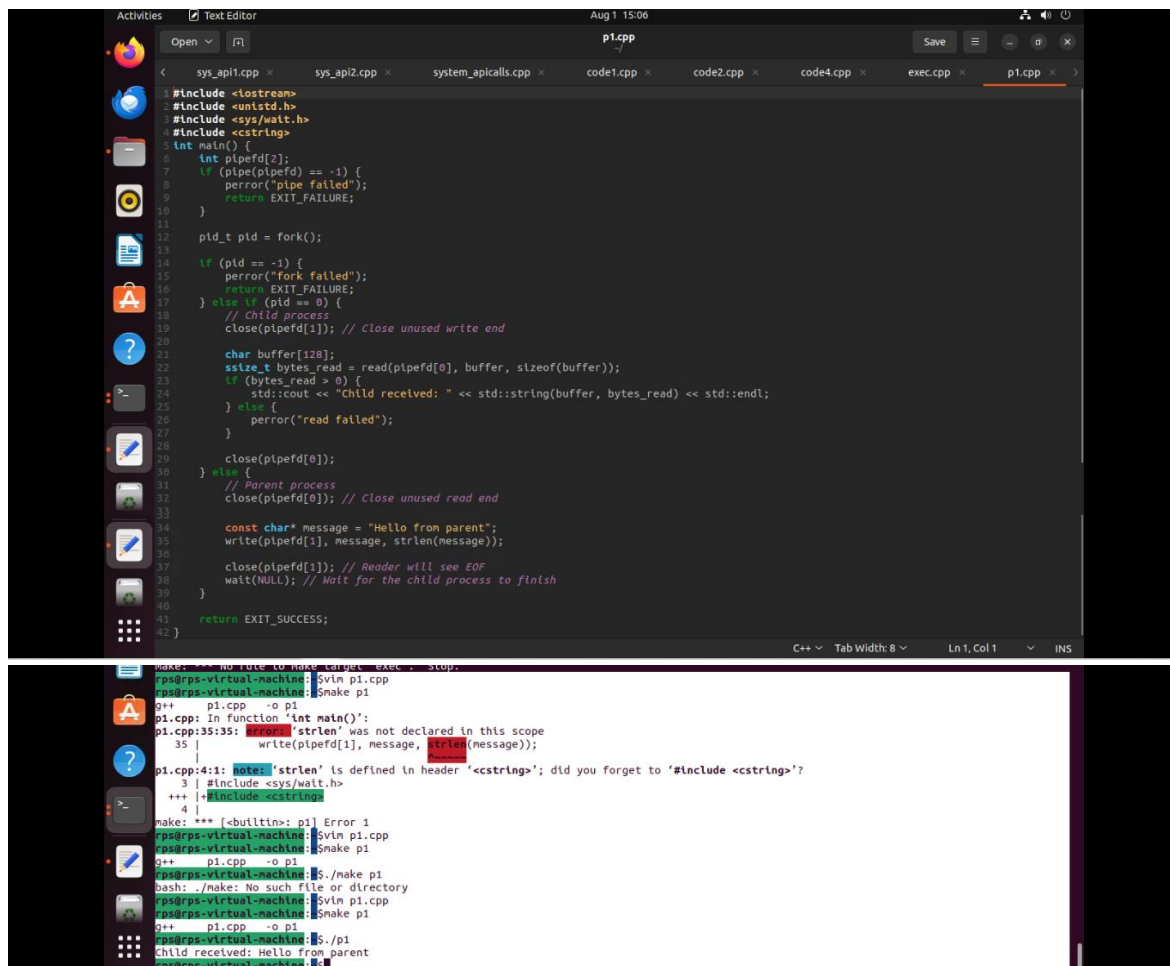
```
ps@rps-virtual-machine:~$ svn code4
ps@rps-virtual-machine:~$ snake code4
g++ code4.cpp -o code4
ps@rps-virtual-machine:~$ ./code4
Child Process
Parent Process
ps@rps-virtual-machine:~$
```

2. Execute a Shell Command:

Write a C++ program that takes a shell command as a string argument and uses exec system calls (e.g., execlp or execv) to execute that command. Handle errors if the command execution fails.

3. Inter-Process Communication with Pipes:

Write a C++ program that demonstrates inter-process communication using pipes. One process should write data to a pipe, and another process should read from the pipe and print the received data. Leverage pipe and fork system calls.



```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>
#include <cstring>

int main() {
    int pipefd[2];
    if (pipe(pipefd) == -1) {
        perror("pipe failed");
        return EXIT_FAILURE;
    }

    pid_t pid = fork();

    if (pid == -1) {
        perror("fork failed");
        return EXIT_FAILURE;
    } else if (pid == 0) {
        // Child process
        close(pipefd[1]); // Close unused write end

        char buffer[128];
        ssize_t bytes_read = read(pipefd[0], buffer, sizeof(buffer));
        if (bytes_read > 0) {
            std::cout << "Child received: " << std::string(buffer, bytes_read) << std::endl;
        } else {
            perror("read failed");
        }

        close(pipefd[0]);
    } else {
        // Parent process
        close(pipefd[0]); // Close unused read end

        const char* message = "Hello from parent";
        write(pipefd[1], message, strlen(message));

        close(pipefd[1]); // Reader will see EOF
        wait(NULL); // Wait for the child process to finish
    }

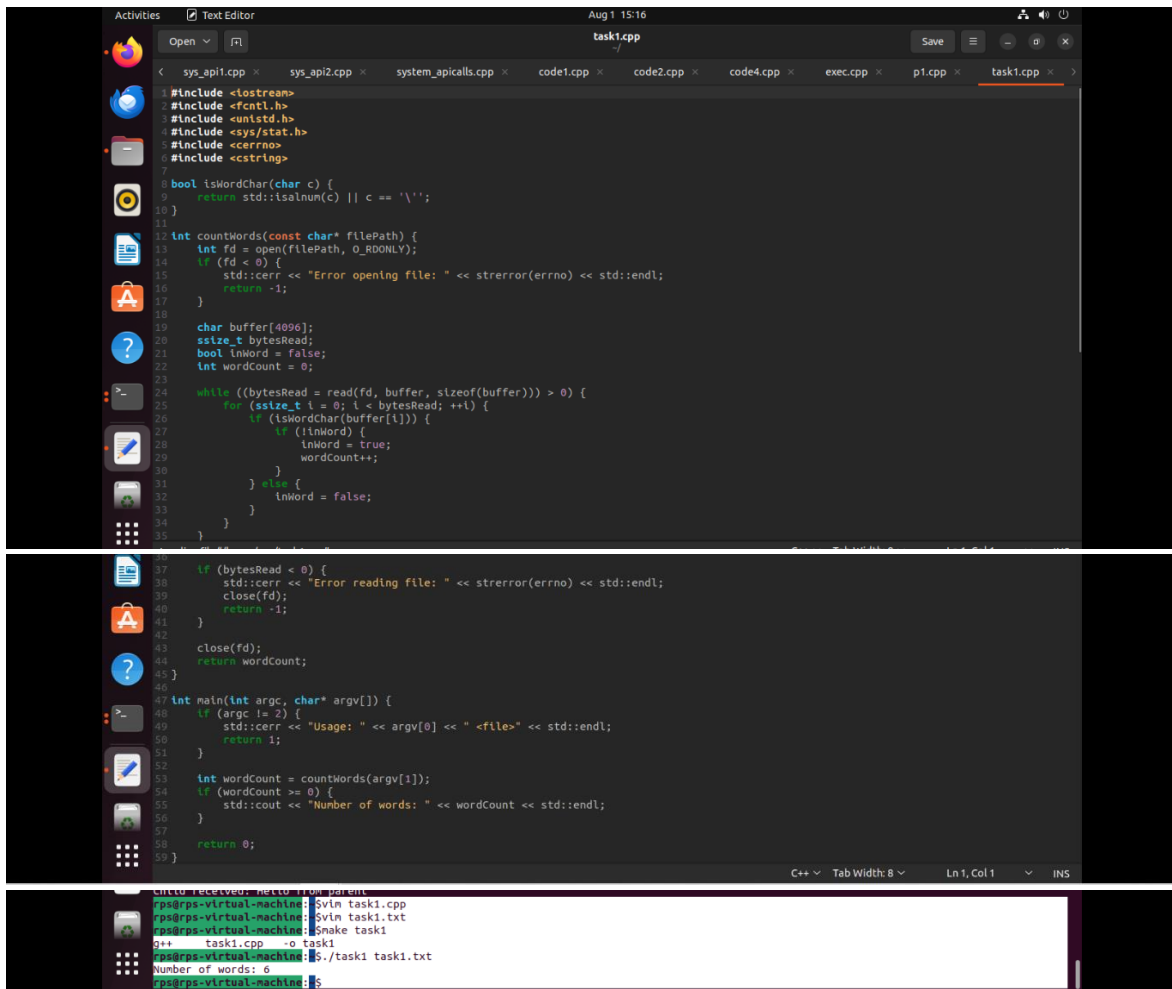
    return EXIT_SUCCESS;
}
```

```
make: *** No rule to make target 'exec'.  Stop.
rps@rps-virtual-machine: $ g++ p1.cpp
rps@rps-virtual-machine: $ ./p1
Child received: Hello from parent
rps@rps-virtual-machine: $
```

C. Text Processing and System Information:

1. Count Words in a File:

Write a C++ program that reads a text file and counts the number of words in it. Use open, read, and close system calls to access the file. Be mindful of delimiters and whitespace characters when counting words.



```
#include <ostream>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <cerrno>
#include <cstring>

bool isWordChar(char c) {
    return std::isalnum(c) || c == '\'';
}

int countWords(const char* filePath) {
    int fd = open(filePath, O_RDONLY);
    if (fd < 0) {
        std::cerr << "Error opening file: " << strerror(errno) << std::endl;
        return -1;
    }

    char buffer[4096];
    ssize_t bytesRead;
    bool inWord = false;
    int wordCount = 0;

    while ((bytesRead = read(fd, buffer, sizeof(buffer))) > 0) {
        for (ssize_t i = 0; i < bytesRead; ++i) {
            if (isWordChar(buffer[i])) {
                if (!inWord) {
                    inWord = true;
                    wordCount++;
                }
            } else {
                inWord = false;
            }
        }
    }

    if (bytesRead < 0) {
        std::cerr << "Error reading file: " << strerror(errno) << std::endl;
        close(fd);
        return -1;
    }
    close(fd);
    return wordCount;
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <file>" << std::endl;
        return 1;
    }

    int wordCount = countWords(argv[1]);
    if (wordCount == 0) {
        std::cout << "Number of words: " << wordCount << std::endl;
    }

    return 0;
}
```

Terminal output:

```
g++ task1.cpp -o task1
./task1 task1.txt
Number of words: 6
```

2. **Get System Uptime:** Write a C++ program that retrieves the system's uptime (time since it was last booted) using appropriate system calls (e.g., getuptime on Linux). Display the uptime information in a user-friendly format.



```
#include <ostream>
#include <cstring>
#include <cerrno>
#include <sys/sysinfo.h>

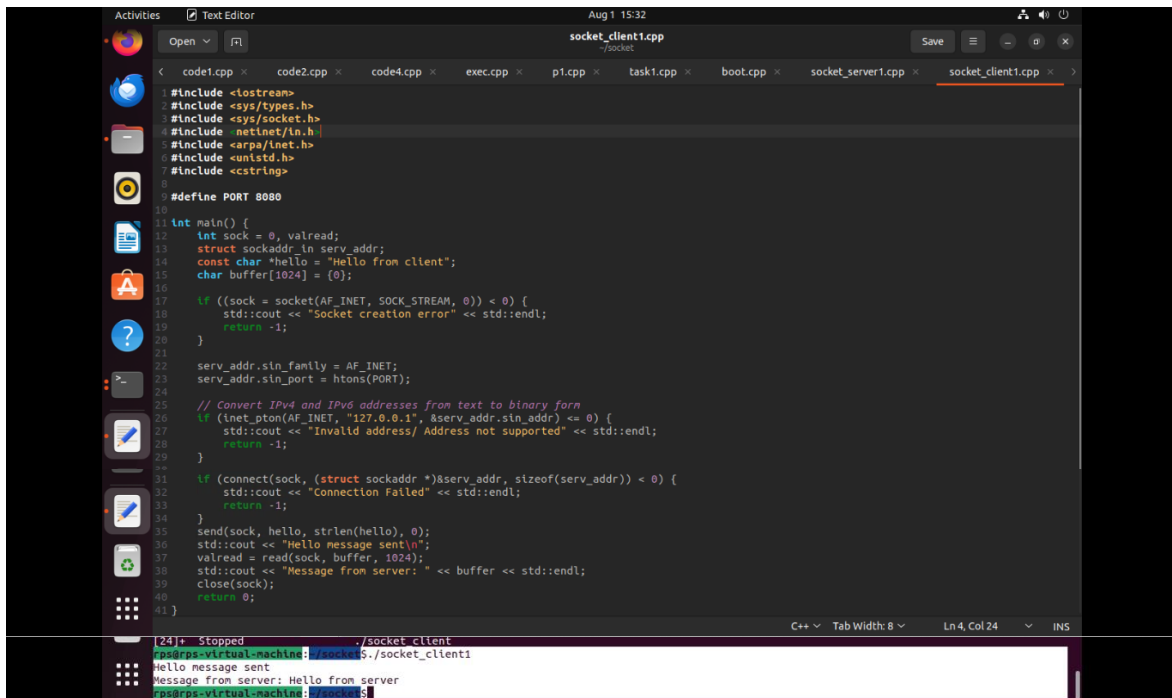
void displayUptime() {
    struct sysinfo info;
    if (sysinfo(&info) != 0) {
        std::cerr << "Error getting system info: " << strerror(errno) << std::endl;
        return;
    }

    long uptime = info.uptime;
    int days = uptime / (24 * 3600);
    uptime %= 24 * 3600;
    int hours = uptime / 3600;
    uptime %= 3600;
    int minutes = uptime / 60;
    int seconds = uptime % 60;

    std::cout << "System Uptime: ";
    if (days > 0) std::cout << days << "d ";
    std::cout << hours << "h " << minutes << "m " << seconds << "s" << std::endl;
}

int main() {
    displayUptime();
    return 0;
}
```


server, receive the server's response, and then close the connection. (Note: Network programming details apply here as well)



The screenshot shows a C++ IDE with a text editor window titled 'socket_client1.cpp'. The code is a client program that connects to a server at 127.0.0.1 on port 8080, sends a 'Hello' message, and receives a response. The output window at the bottom shows the program's execution, including the message received from the server.

```
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>

#define PORT 8080

int main() {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    const char *hello = "Hello from client";
    char buffer[1024] = {0};

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cout << "Socket creation error" << std::endl;
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        std::cout << "Invalid address/ Address not supported" << std::endl;
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cout << "Connection Failed" << std::endl;
        return -1;
    }

    send(sock, hello, strlen(hello), 0);
    std::cout << "Hello message sent\n";
    valread = read(sock, buffer, 1024);
    std::cout << "Message from server: " << buffer << std::endl;
    close(sock);
    return 0;
}
```

Output:

```
[24]+ Stopped ./socket_client
rpsdrps-virtual-machine:~/socket1$ ./socket_client1
Hello message sent
Message from server: Hello from server
rpsdrps-virtual-machine:~/socket1$
```