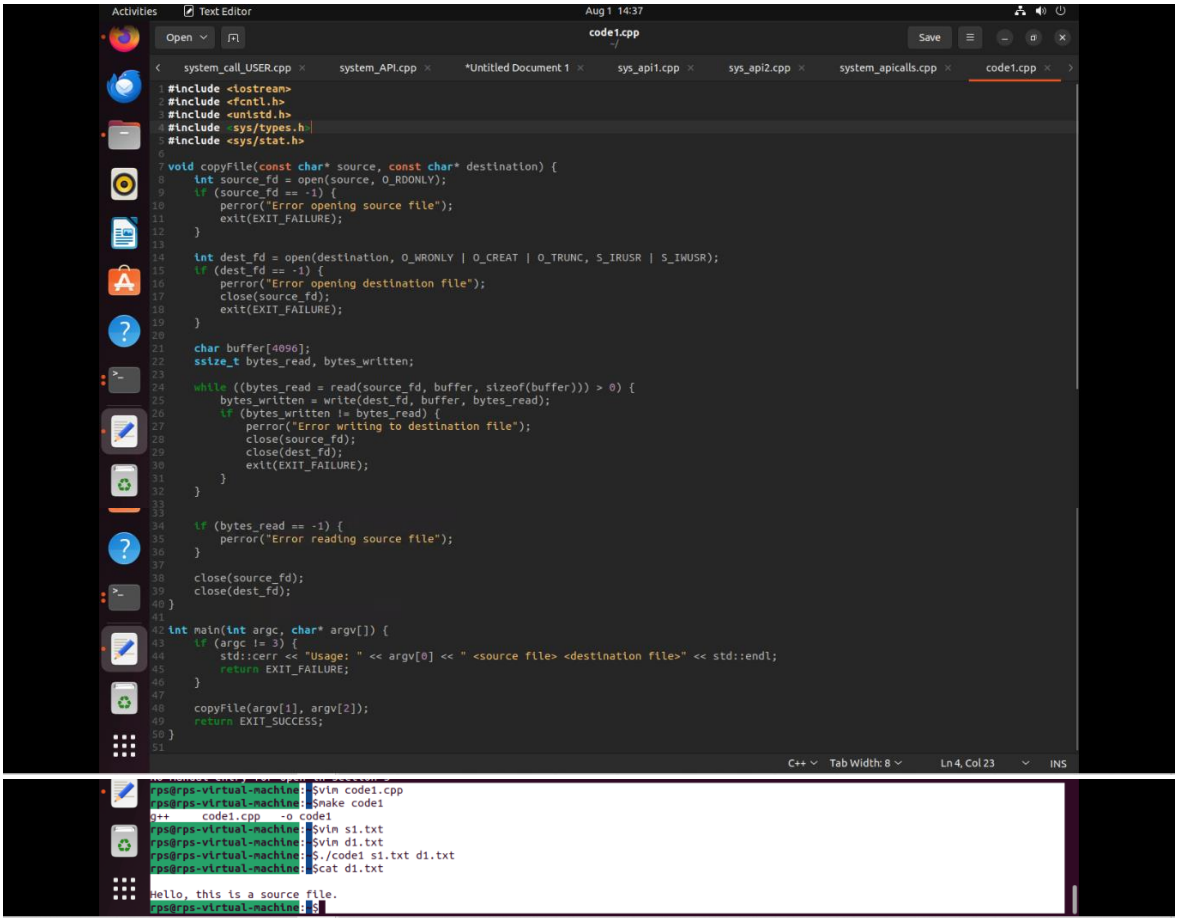
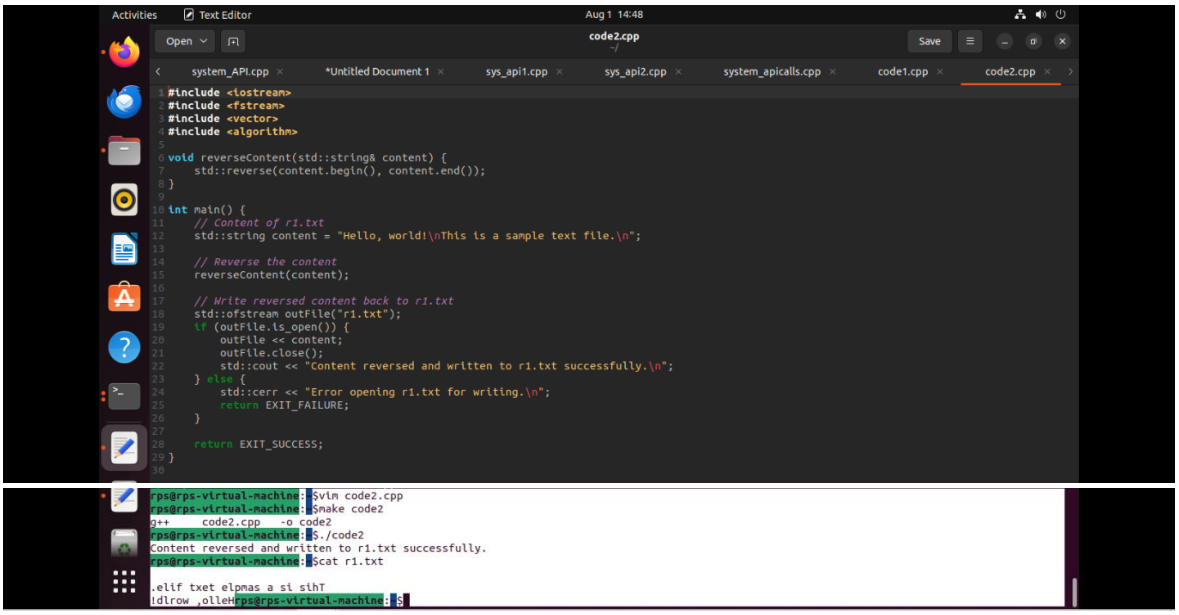
Day -12 LSP Assignment (Task -2)

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).



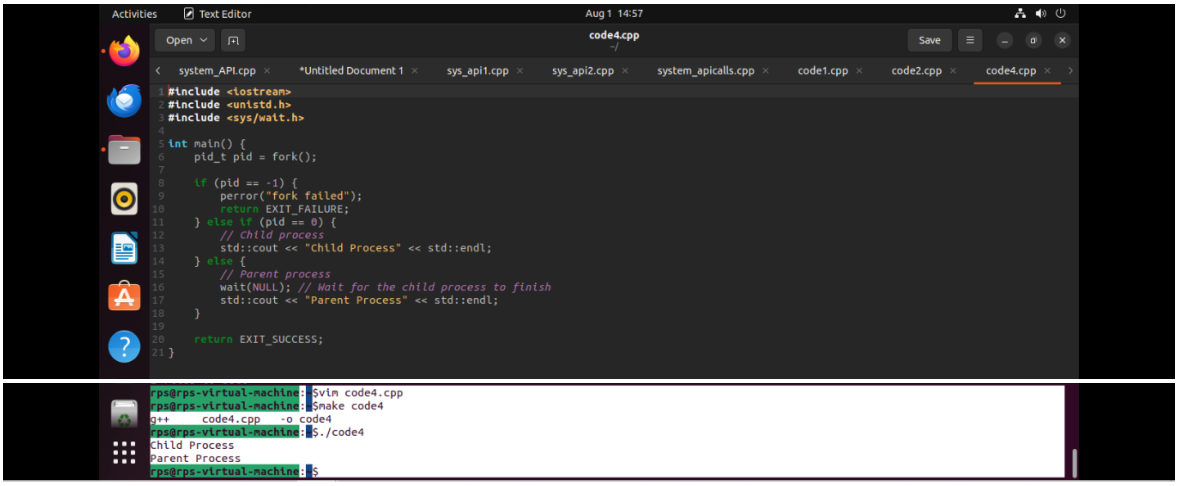
2. **Reverse a File:**
   Write a C++ program that reads the contents of a file line by line, reverses each line in-place, and then writes the reversed lines back to the same file. Use system calls like open, read, write, lseek, and close to achieve this.



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.

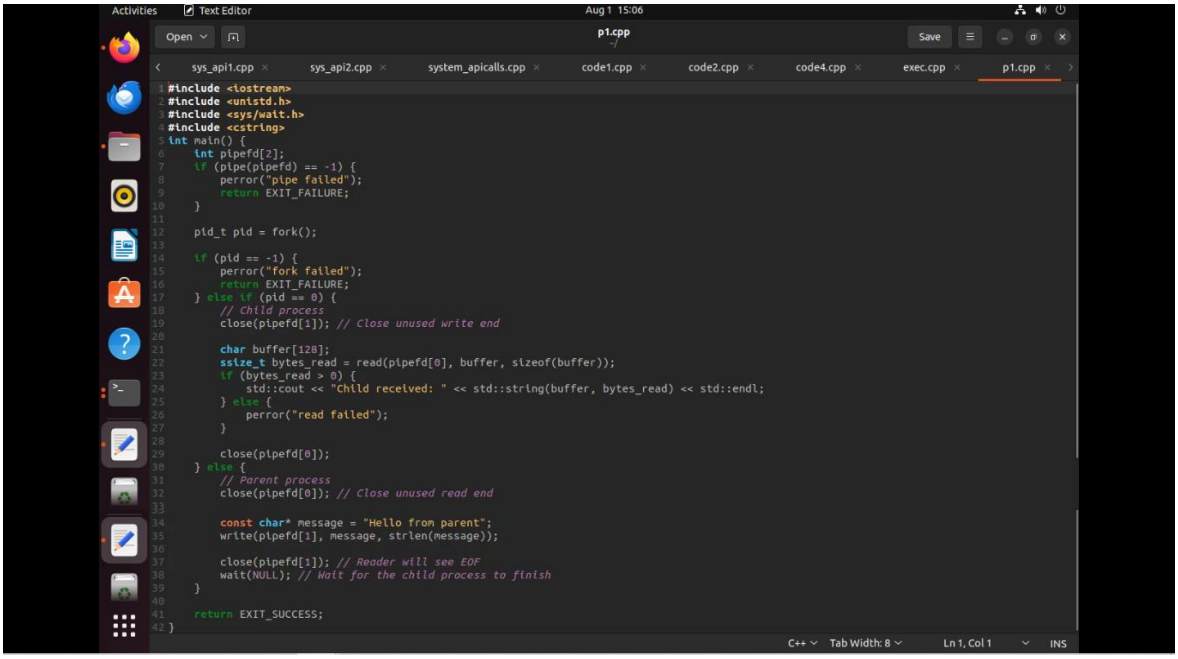


2. Execute a Shell Command:

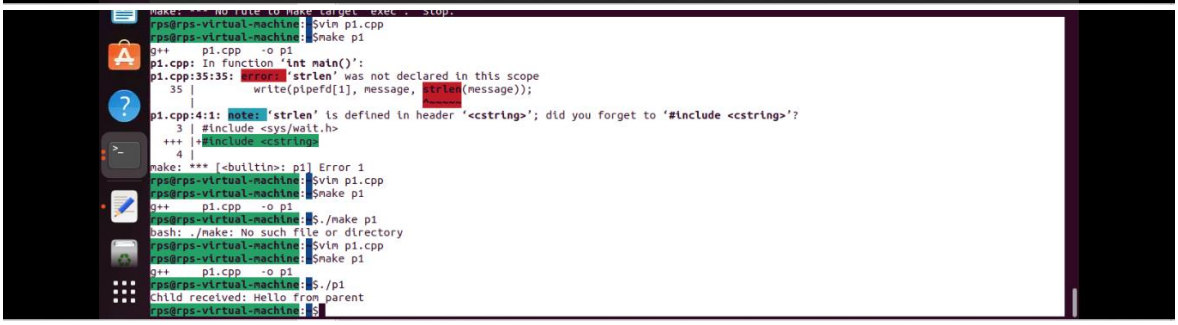
Write a C++ program that takes a shell command as a string argument and uses exec system calls (e.g., execlp or execv) to execute that command. Handle errors if the command execution fails.

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;
}
```

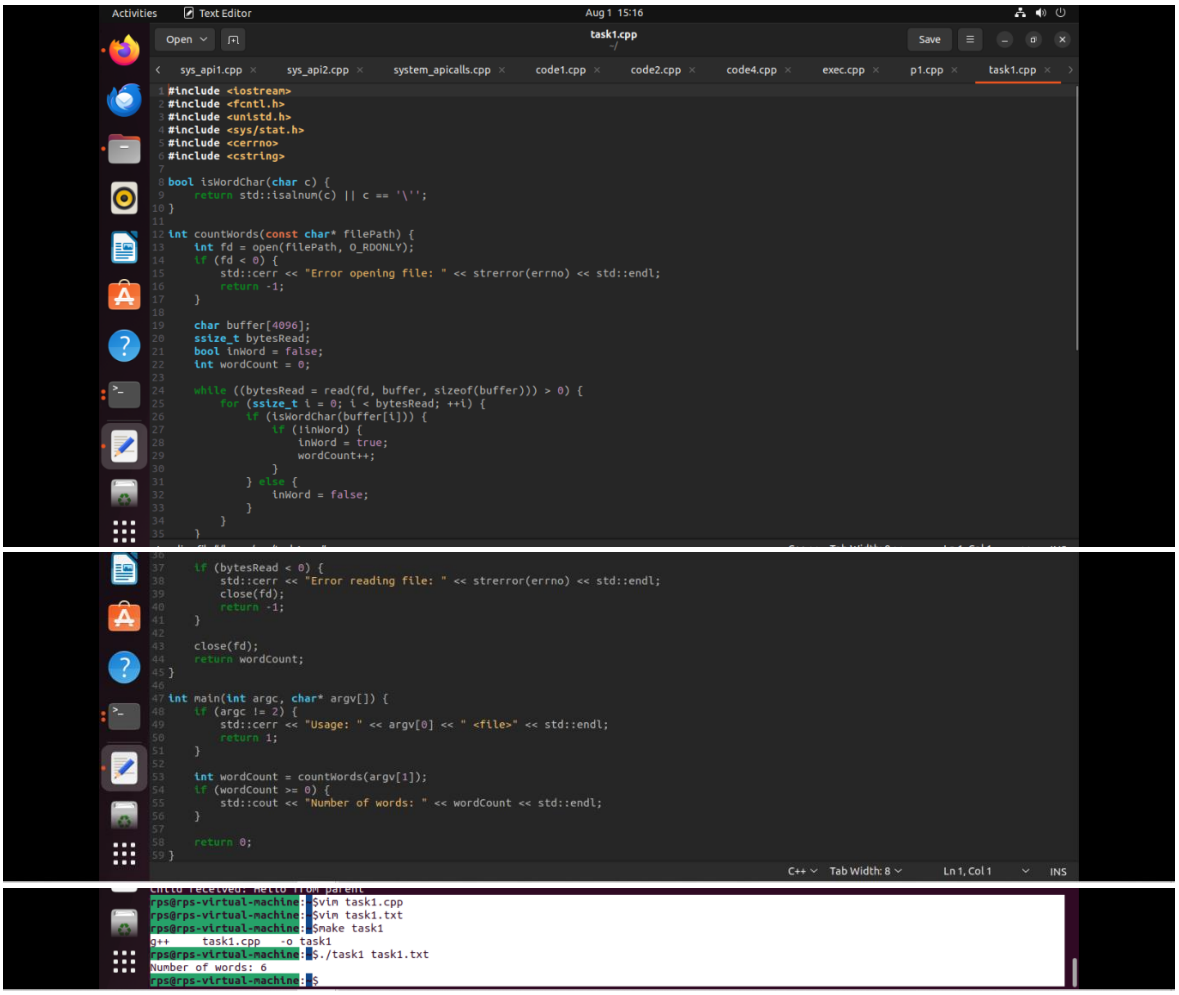


```
rps@rps-virtual-machine:~$vim p1.cpp
rps@rps-virtual-machine:~$make p1
g++     p1.cpp   -o p1
p1.cpp: In function 'int main()':
p1.cpp:35:35: error: 'strlen' was not declared in this scope
   35 |         write(pipefd[1], message, strlen(message));
      |                                   ^~~~~~
p1.cpp:4:1: note: 'strlen' is defined in header '<cstring>'; did you forget to '#include <cstring>'?
    3 | #include <sys/wait.h>
  +++ |+#include <cstring>
    4 |
make: *** [<builtin>: p1] Error 1
rps@rps-virtual-machine:~$vim p1.cpp
rps@rps-virtual-machine:~$make p1
g++     p1.cpp   -o p1
rps@rps-virtual-machine:~$./make p1
bash: ./make: No such file or directory
rps@rps-virtual-machine:~$vim p1.cpp
rps@rps-virtual-machine:~$make p1
g++     p1.cpp   -o p1
rps@rps-virtual-machine:~$./p1
Child received: Hello from parent
rps@rps-virtual-machine:~$
```

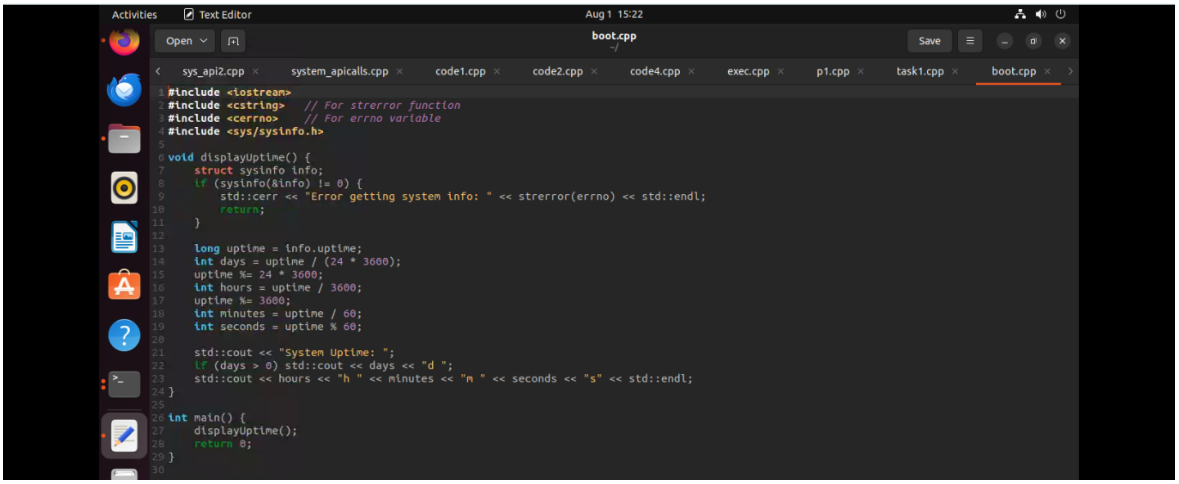
C. Text Processing and System Information:

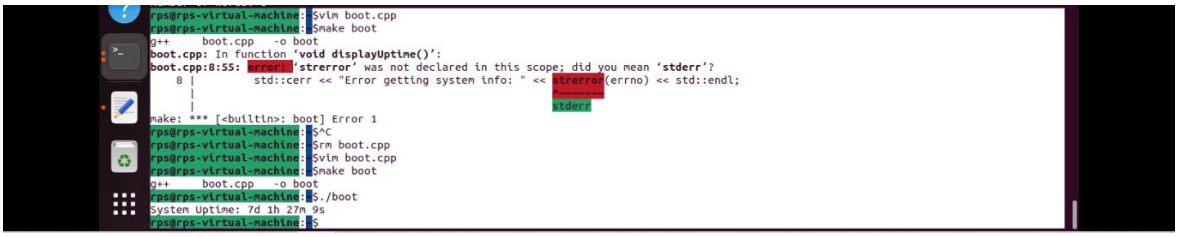
1. **Count Words in a File:**

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.



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.

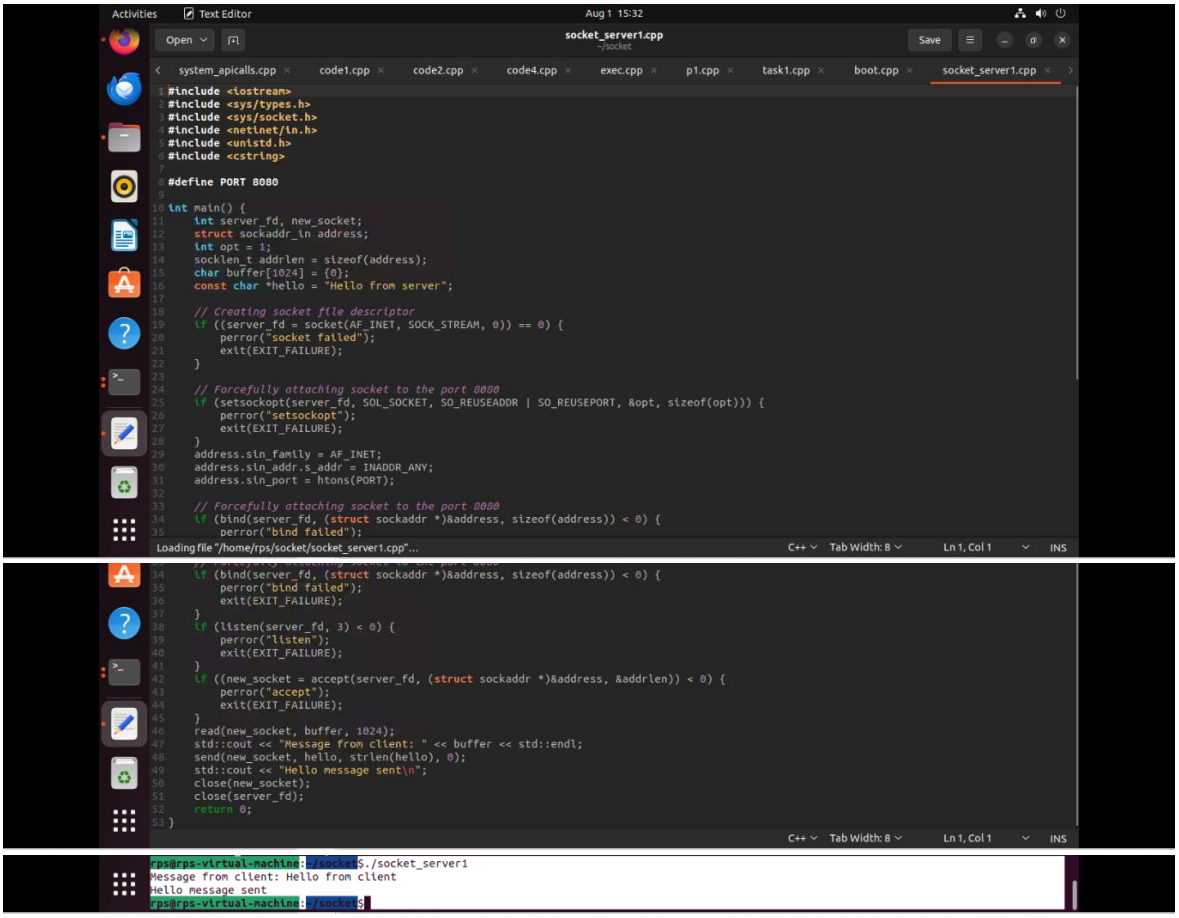




D. Network Programming (Bonus):

1. Simple TCP Server:

Write a basic C++ program that acts as a server that listens for incoming TCP connections on a specific port. Upon receiving a connection, the server should send a predefined message (e.g., "Hello, client!") to the client and then close the connection. Utilize system calls like socket, bind, listen, accept, send, and recv. (Note: Network programming involves additional libraries/headers. Refer to system documentation)



2. Simple TCP Client:

As a companion to question 9, write a C++ program that acts as a client that connects to the server created in question 9. The client should send a message (e.g., "Hi from client!") to the

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