Semin Bae

Tony Mione

CSE320 / Fall 2024

Nov/19/2024

Report_HW6

Summarizing everything such as each step or functions that you implemented

Server:

1) Initializes a file descriptor set readfds to monitor the client socket (fd) and the server's standard input (STDIN_FILENO).
2) Enters a loop using select() to wait for input from either the client or the server's keyboard.
3) When Client send data to Server
    a) Reads data from the client using readline()
    b) Checks for termination conditions whether Client send ‘exit’ or len > 50 line
    c) if yes, Exits the child process and send SIGTERM to terminate parent’s process using kill(getppid(), SIGTERM)
    d) If not, the server prints the message received from the client.
4) When Server’s standard input is ready(Server send Data to Client)
    a) Reads input from the keyboard using fgets().
    b) Checks for termination conditions whether Client send ‘exit’ or len > 50 line
        i) if yes, Exits the child process and send SIGTERM to terminate parent’s process using kill(getppid(), SIGTERM)
        ii) Sends the message to the client using writen()
5) If had signal SIGTERM, signal call the custom signal handler handle_sigterm to terminate program properly

Client:

1) Initializes a file descriptor set readfds to monitor the server socket (sfd) and the client's standard input (STDIN_FILENO).
2) Enters a loop using select() to wait for input from either the server or the client's keyboard.
3) When server sends data to Client
    a) Reads data from the server using readline().
    b) Checks if the server closed the connection.
    c) Prints the message received from the server.
4) When Client’s standard input is ready (Client send data to Server)
    a) if len(line) > 50 or line == ‘exit’ then print log its log
    b) send the message to the server using writen().

Challenges faced

- Initially, I faced a challenge where only the child process would terminate, leaving the parent process running and preventing the server from shutting down completely. To resolve this, I modified the code so that when the child process detects a termination condition (type exit, length over 50), it sends a SIGTERM signal to the parent process. The parent then uses a custom signal handler to perform cleanup and terminate gracefully. This ensured that both the child and parent processes terminate properly, allowing the server to shut down completely.

What I've learned

- I learned and used some socket connection related methods. It was a good experience to use methods not only studied in the book.

1) CHKBOOLQUIT(condition, message);: A macro that checks a condition, and if the condition is false, prints an error message and exits the program.
2) FD_ZERO(set);: Clears all entries from a file descriptor set.
3) FD_SET(descriptor, set);: Adds a file descriptor to a file descriptor set.
4) FD_ISSET(descriptor, set);: Checks if a file descriptor is included in a file descriptor set.
5) writen(descriptor, buffer, nbytes);: Writes a specified number of bytes from a buffer to a file descriptor, ensuring all bytes are written.
6) select(nfds, readfds, writefds, exceptfds, timeout);: Waits for one or more file descriptors to become ready for I/O operations.