NETWORK PROGRAMMING REPORT
Comparison of Iterative and Concurrent Servers

1. Introduction
In this report, we compare iterative and concurrent servers, analyzing their advantages, disadvantages, real-world applications, and system resource usage.

2. Iterative Server
- Handles one client at a time.
- Processes the client request, then closes the connection before accepting another.
- Simple to implement but slow for handling multiple clients.

Advantages:
- Low resource usage.
- Easy to develop and debug.

Disadvantages:
- Poor scalability (blocks new clients until the current one finishes).
- Not suitable for real-time applications.

Real-World Applications:
- FTP servers (single-user mode).
- Debugging or testing environments.

3. Concurrent Server
- Can handle multiple clients at the same time using threads or processes.
- More efficient for large-scale applications.

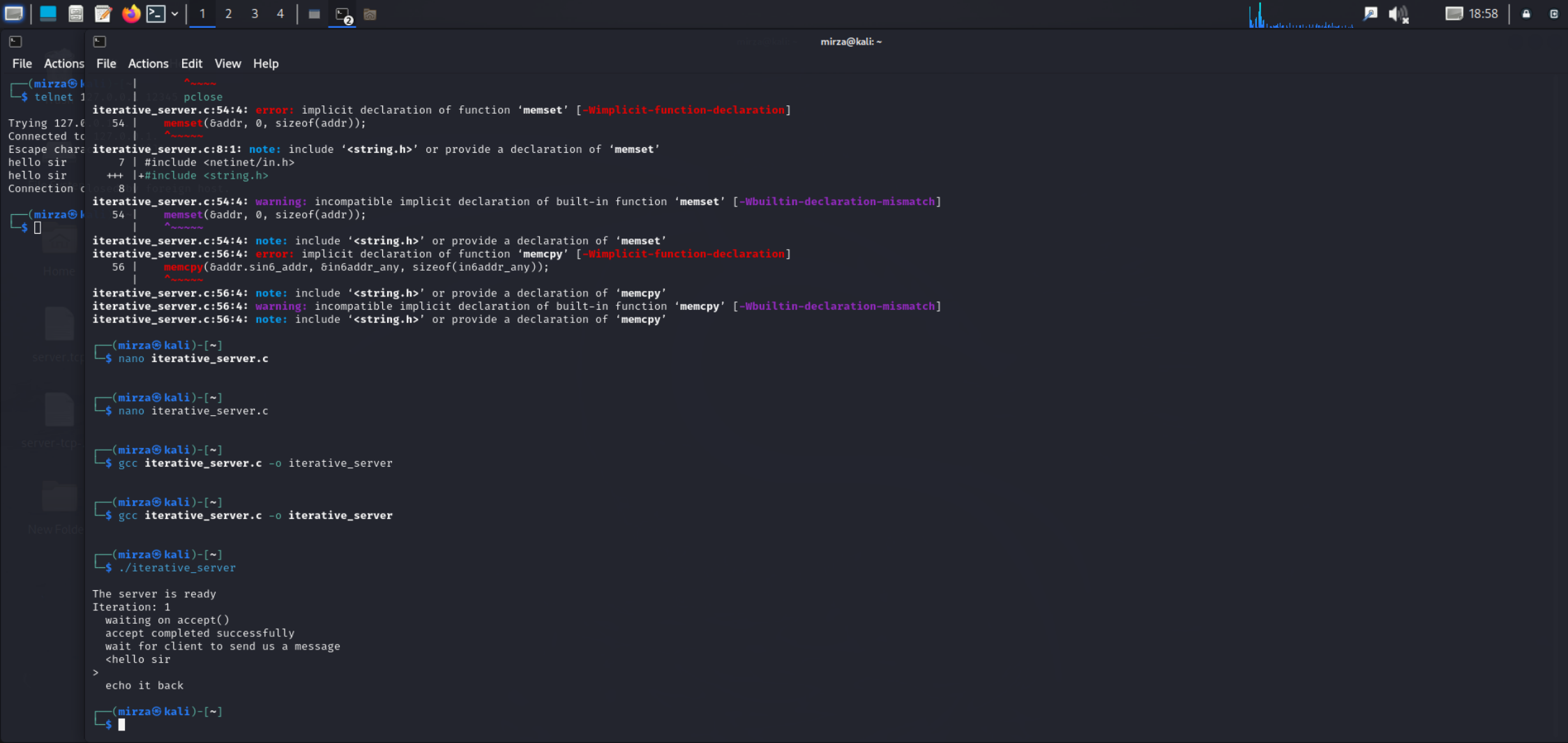
Advantages:
- Fast response time.
- Can serve many clients simultaneously.
- Ideal for high-traffic applications.

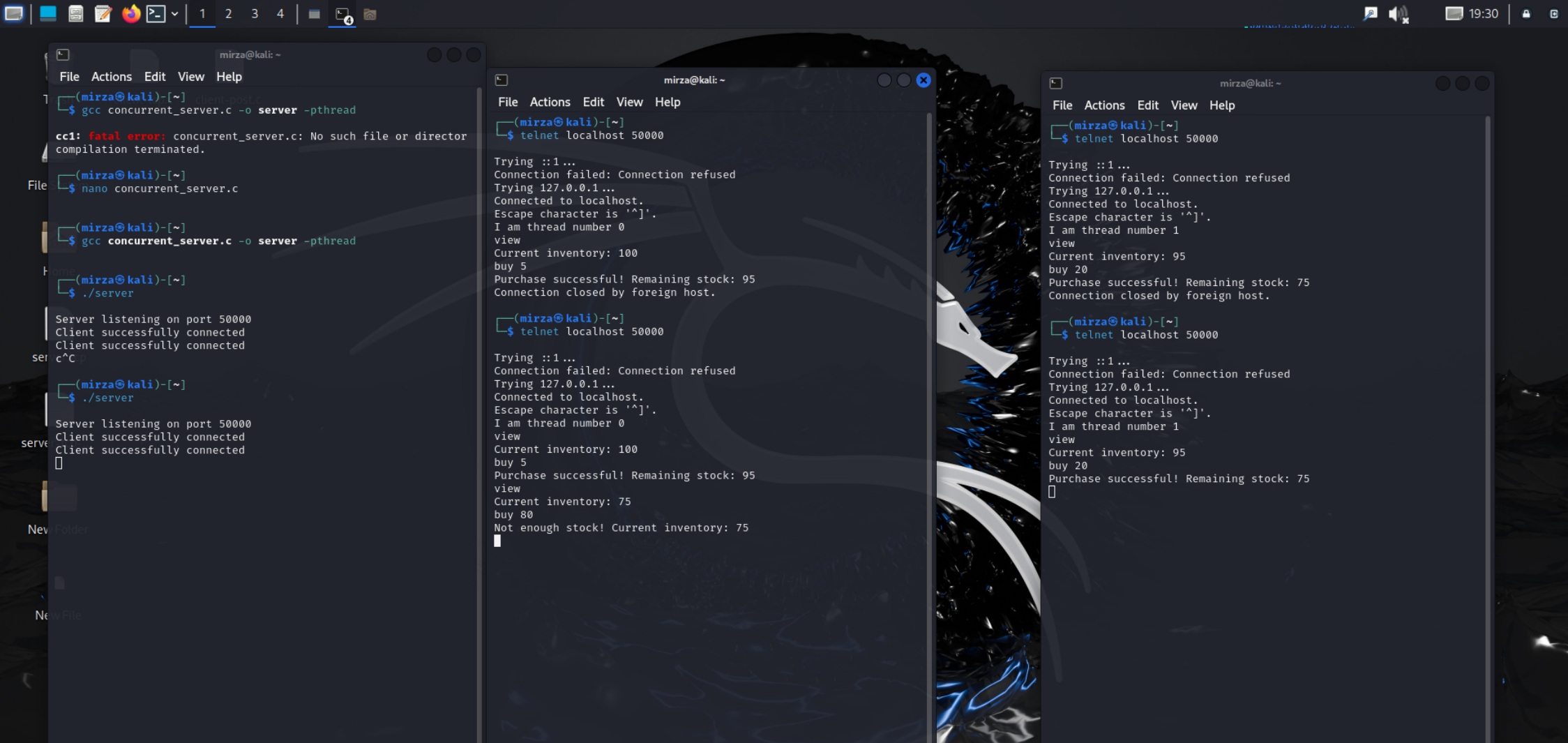
Disadvantages:
- Higher resource consumption.
- More complex to implement due to thread synchronization issues.

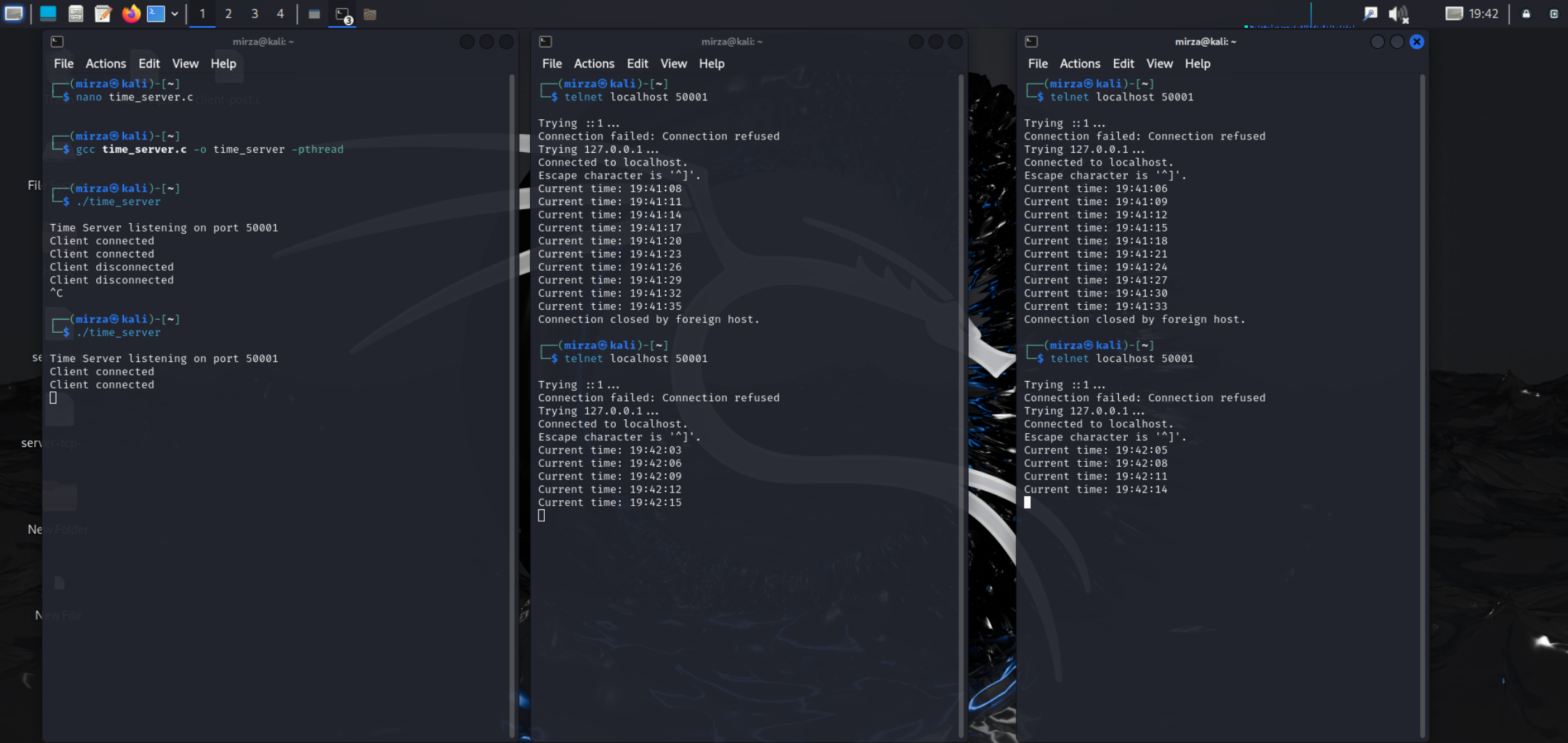
Real-World Applications:
- Web servers (Apache, Nginx).
- Online multiplayer games.
- Real-time stock market systems.

4. System Resource Usage Analysis
- Iterative servers use minimal resources but are slow under heavy load.
- Concurrent servers handle high loads but consume more CPU and memory.
- System resource usage can be monitored using the command:
  `ps aux | grep server`

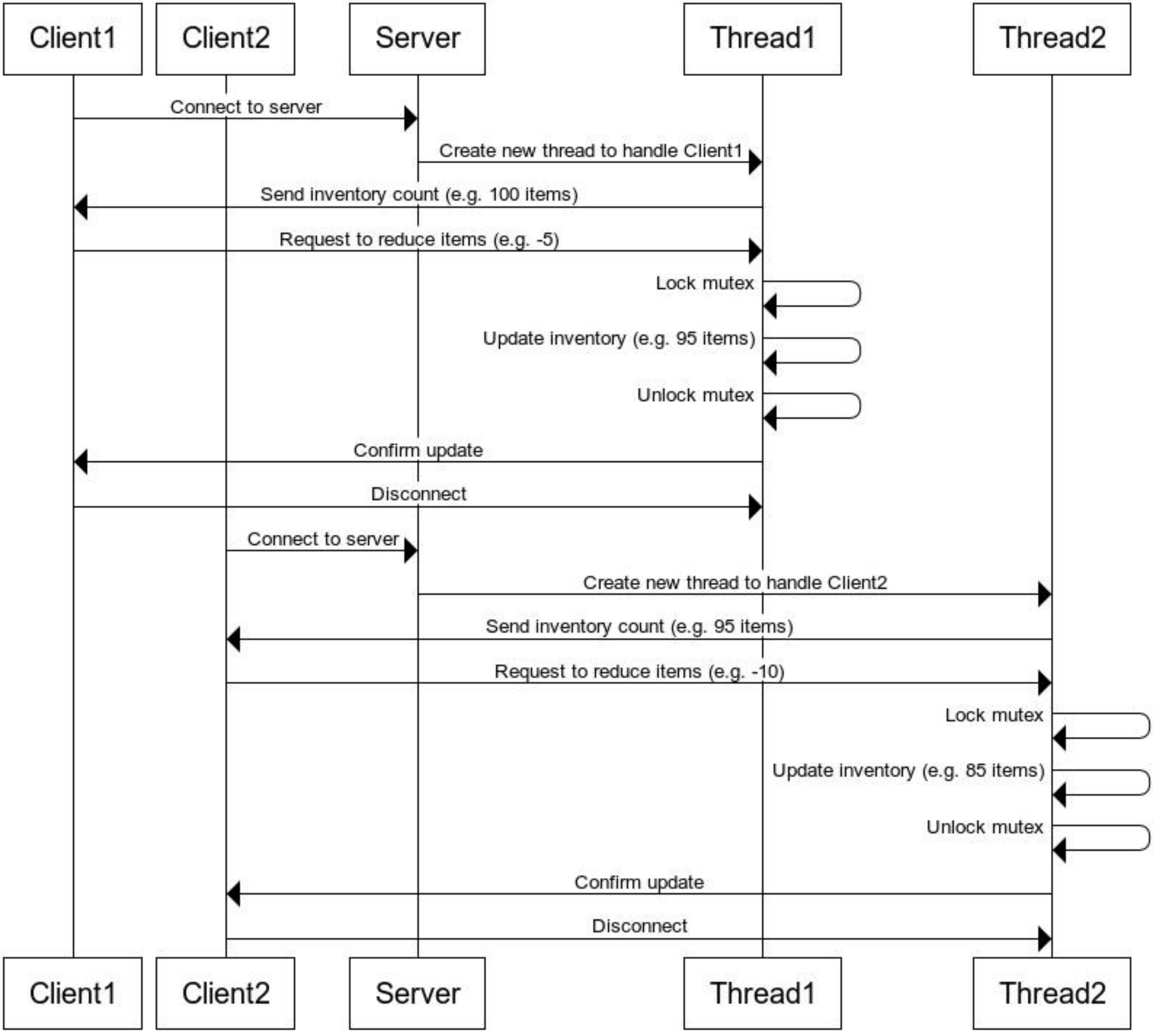
5. Conclusion
- Iterative servers are useful for small applications but lack scalability.
- Concurrent servers are preferred for modern applications requiring multiple connections.


File Actions Edit View Help
mirza@kali: ~
File Actions Edit View Help
| ^~~~~
| pclose
iterative_server.c:54:4: error: implicit declaration of function ‘memset’ [-Wimplicit-function-declaration]
54 | memset(&addr, 0, sizeof(addr));
| ^~~~~
iterative_server.c:8:1: note: include ‘<string.h>’ or provide a declaration of ‘memset’
7 | #include <netinet/in.h>
+++ |+#include <string.h>
8 |
iterative_server.c:54:4: warning: incompatible implicit declaration of built-in function ‘memset’ [-Wbuiltin-declaration-mismatch]
54 | memset(&addr, 0, sizeof(addr));
| ^~~~~
iterative_server.c:54:4: note: include ‘<string.h>’ or provide a declaration of ‘memset’
iterative_server.c:56:4: error: implicit declaration of function ‘memcpy’ [-Wimplicit-function-declaration]
56 | memcpy(&addr.sin6_addr, &in6addr_any, sizeof(in6addr_any));
| ^~~~~
iterative_server.c:56:4: note: include ‘<string.h>’ or provide a declaration of ‘memcpy’
iterative_server.c:56:4: warning: incompatible implicit declaration of built-in function ‘memcpy’ [-Wbuiltin-declaration-mismatch]
iterative_server.c:56:4: note: include ‘<string.h>’ or provide a declaration of ‘memcpy’
(mirza㉿kali)-[~]
$ nano iterative_server.c
(mirza㉿kali)-[~]
$ nano iterative_server.c
(mirza㉿kali)-[~]
$ gcc iterative_server.c -o iterative_server
(mirza㉿kali)-[~]
$ gcc iterative_server.c -o iterative_server
(mirza㉿kali)-[~]
$ ./iterative_server
The server is ready
Iteration: 1
waiting on accept()
accept completed successfully
wait for client to send us a message
<hello sir
>
echo it back
(mirza㉿kali)-[~]
$

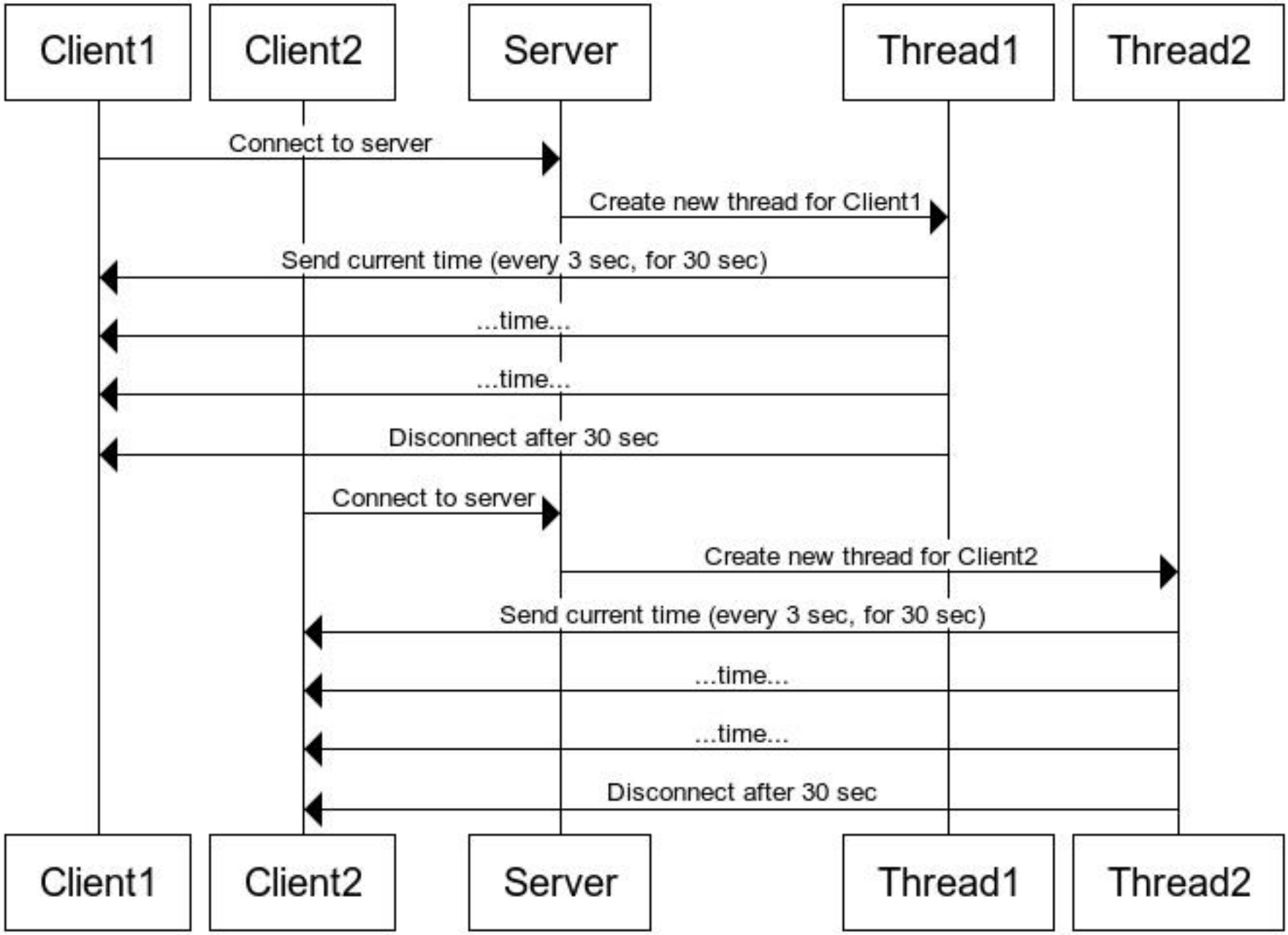

mirza@kali: ~
File Actions Edit View Help
(mirza㉿kali)-[~]
$ gcc concurrent_server.c -o server -pthread
cc1: fatal error: concurrent_server.c: No such file or director
compilation terminated.
(mirza㉿kali)-[~]
$ nano concurrent_server.c
(mirza㉿kali)-[~]
$ gcc concurrent_server.c -o server -pthread
(mirza㉿kali)-[~]
$ ./server
Server listening on port 50000
Client successfully connected
Client successfully connected
c^C
(mirza㉿kali)-[~]
$ ./server
Server listening on port 50000
Client successfully connected
Client successfully connected
mirza@kali: ~
File Actions Edit View Help
(mirza㉿kali)-[~]
$ telnet localhost 50000
Trying ::1 ...
Connection failed: Connection refused
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
I am thread number 0
view
Current inventory: 100
buy 5
Purchase successful! Remaining stock: 95
Connection closed by foreign host.
(mirza㉿kali)-[~]
$ telnet localhost 50000
Trying ::1 ...
Connection failed: Connection refused
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
I am thread number 0
view
Current inventory: 100
buy 5
Purchase successful! Remaining stock: 95
view
Current inventory: 75
buy 80
Not enough stock! Current inventory: 75
mirza@kali: ~
File Actions Edit View Help
(mirza㉿kali)-[~]
$ telnet localhost 50000
Trying ::1 ...
Connection failed: Connection refused
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
I am thread number 1
view
Current inventory: 95
buy 20
Purchase successful! Remaining stock: 75
Connection closed by foreign host.
(mirza㉿kali)-[~]
$ telnet localhost 50000
Trying ::1 ...
Connection failed: Connection refused
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
I am thread number 1
view
Current inventory: 95
buy 20
Purchase successful! Remaining stock: 75


mirza@kali: ~
File Actions Edit View Help
(mirza㉿kali)-[~]
$ nano time_server.c
(mirza㉿kali)-[~]
$ gcc time_server.c -o time_server -pthread
(mirza㉿kali)-[~]
$ ./time_server
Time Server listening on port 50001
Client connected
Client connected
Client disconnected
Client disconnected
^C
(mirza㉿kali)-[~]
$ ./time_server
Time Server listening on port 50001
Client connected
Client connected
mirza@kali: ~
File Actions Edit View Help
(mirza㉿kali)-[~]
$ telnet localhost 50001
Trying ::1 ...
Connection failed: Connection refused
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
Current time: 19:41:08
Current time: 19:41:11
Current time: 19:41:14
Current time: 19:41:17
Current time: 19:41:20
Current time: 19:41:23
Current time: 19:41:26
Current time: 19:41:29
Current time: 19:41:32
Current time: 19:41:35
Connection closed by foreign host.
(mirza㉿kali)-[~]
$ telnet localhost 50001
Trying ::1 ...
Connection failed: Connection refused
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
Current time: 19:42:03
Current time: 19:42:06
Current time: 19:42:09
Current time: 19:42:12
Current time: 19:42:15
mirza@kali: ~
File Actions Edit View Help
(mirza㉿kali)-[~]
$ telnet localhost 50001
Trying ::1 ...
Connection failed: Connection refused
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
Current time: 19:41:06
Current time: 19:41:09
Current time: 19:41:12
Current time: 19:41:15
Current time: 19:41:18
Current time: 19:41:21
Current time: 19:41:24
Current time: 19:41:27
Current time: 19:41:30
Current time: 19:41:33
Connection closed by foreign host.
(mirza㉿kali)-[~]
$ telnet localhost 50001
Trying ::1 ...
Connection failed: Connection refused
Trying 127.0.0.1 ...
Connected to localhost.
Escape character is '^]'.
Current time: 19:42:05
Current time: 19:42:08
Current time: 19:42:11
Current time: 19:42:14

Concurrent Server - Stockroom Inventory System



Concurrent Time Server - 7 Observers



concurrent_server.c

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include <string.h>
 4: #include <unistd.h>
 5: #include <sys/socket.h>
 6: #include <netinet/in.h>
 7: #include <pthread.h>
 8:
 9: #define SERVER_PORT 50000
10: #define MAX_CLIENTS 8
11:
12: int inventory = 100; // Shared inventory count
13: pthread_mutex_t lock; // Mutex for thread safety
14:
15: struct thread_arguments {
16:     int number;
17:     int accept_sd;
18: };
19:
20: void *thread_function(void *arguments);
21:
22: int main(void) {
23:     int listen_sd, accept_sd, on = 1, thread_number = 0;
24:     struct sockaddr_in addr;
25:     pthread_t thread_id;
26:
27:     // Initialize the mutex
28:     pthread_mutex_init(&lock, NULL);
29:
30:     // Create the server socket
31:     listen_sd = socket(AF_INET, SOCK_STREAM, 0);
32:     setsockopt(listen_sd, SOL_SOCKET, SO_REUSEADDR, (char *)&on, sizeof(on));
33:
34:     // Configure the server address
35:     memset(&addr, 0, sizeof(addr));
36:     addr.sin_family = AF_INET;
37:     addr.sin_addr.s_addr = htonl(INADDR_ANY);
38:     addr.sin_port = htons(SERVER_PORT);
39:     bind(listen_sd, (struct sockaddr *)&addr, sizeof(addr));
40:
41:     // Start listening for incoming connections
42:     listen(listen_sd, MAX_CLIENTS);
43:     printf("Server listening on port %d\n", SERVER_PORT);
44:
45:     while (1) {
46:         accept_sd = accept(listen_sd, NULL, NULL);
47:         printf("Client successfully connected\n");
48:
49:         // Allocate memory for thread arguments
50:         struct thread_arguments *thread_arg = malloc(sizeof(struct thread_arguments));
51:         thread_arg->number = thread_number;
52:         thread_arg->accept_sd = accept_sd;
```

```
53:
54:         // Create a new thread for the client
55:         pthread_create(&thread_id, NULL, thread_function, (void *)thread_arg);
56:         pthread_detach(thread_id); // Automatically free thread resources
57:
58:         thread_number++;
59:     }
60:
61:     // Clean up resources
62:     pthread_mutex_destroy(&lock);
63:     close(listen_sd);
64:     return 0;
65: }
66:
67: void *thread_function(void *arguments) {
68:     struct thread_arguments *p = (struct thread_arguments *)arguments;
69:     int accept_sd = p->accept_sd;
70:     int thread_number = p->number;
71:     free(p); // Free allocated memory
72:
73:     char buffer[80], response[80];
74:     int read_size, quantity;
75:
76:     // Send thread information to client
77:     sprintf(response, "I am thread number %d\n", thread_number);
78:     send(accept_sd, response, strlen(response), 0);
79:
80:     while ((read_size = recv(accept_sd, buffer, sizeof(buffer) - 1, 0)) > 0) {
81:         buffer[read_size] = '\0';
82:
83:         if (strncmp(buffer, "view", 4) == 0) {
84:             pthread_mutex_lock(&lock);
85:             sprintf(response, "Current inventory: %d\n", inventory);
86:             pthread_mutex_unlock(&lock);
87:         } else if (strncmp(buffer, "buy", 3) == 0) {
88:             // Extract quantity from "buy X" command
89:             if (sscanf(buffer, "buy %d", &quantity) == 1) {
90:                 pthread_mutex_lock(&lock);
91:                 if (quantity > 0 && inventory >= quantity) {
92:                     inventory -= quantity;
93:                     sprintf(response, "Purchase successful! Remaining stock: %d\n", inve
94:                 } else {
95:                     sprintf(response, "Not enough stock! Current inventory: %d\n", inven
96:                 }
97:                 pthread_mutex_unlock(&lock);
98:             } else {
99:                 sprintf(response, "Invalid format! Use: buy <amount>\n");
100:             }
101:         } else {
102:             sprintf(response, "Unknown command. Use 'view' or 'buy <amount>'.\n");
103:         }
104:         send(accept_sd, response, strlen(response), 0);
105:     }
106:
```

```
107:     // Close client socket
108:     close(accept_sd);
109:     pthread_exit(NULL);
110: }
```

time_server.c

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: #include <string.h>
 4: #include <unistd.h>
 5: #include <sys/socket.h>
 6: #include <netinet/in.h>
 7: #include <pthread.h>
 8: #include <time.h>
 9:
10: #define SERVER_PORT 50001  // Port number for the Time Server
11: #define MAX_CLIENTS 7      // Maximum 7 clients
12: #define RUN_TIME 30        // Clients stay connected for 30 seconds
13: #define INTERVAL 3         // Time update every 3 seconds
14:
15: struct thread_arguments {
16:     int accept_sd;
17: };
18:
19: // Function to send time updates
20: void *thread_function(void *arguments);
21:
22: int main(void) {
23:     int listen_sd, accept_sd, on = 1;
24:     struct sockaddr_in addr;
25:     pthread_t thread_id;
26:
27:     // Create a TCP socket
28:     listen_sd = socket(AF_INET, SOCK_STREAM, 0);
29:     setsockopt(listen_sd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
30:
31:     // Configure the server address
32:     memset(&addr, 0, sizeof(addr));
33:     addr.sin_family = AF_INET;
34:     addr.sin_addr.s_addr = htonl(INADDR_ANY);
35:     addr.sin_port = htons(SERVER_PORT);
36:     bind(listen_sd, (struct sockaddr *)&addr, sizeof(addr));
37:
38:     // Start listening for client connections
39:     listen(listen_sd, MAX_CLIENTS);
40:     printf("Time Server listening on port %d\n", SERVER_PORT);
41:
42:     while (1) {
43:         accept_sd = accept(listen_sd, NULL, NULL);
44:         printf("Client connected\n");
45:
46:         // Allocate memory for thread arguments
47:         struct thread_arguments *thread_arg = malloc(sizeof(struct thread_arguments));
48:         thread_arg->accept_sd = accept_sd;
49:
50:         // Create a thread for each client
51:         pthread_create(&thread_id, NULL, thread_function, (void *)thread_arg);
52:         pthread_detach(thread_id); // Automatically free thread resources
```

```
53:     }
54:
55:     // Close the listening socket (not reached in normal operation)
56:     close(listen_sd);
57:     return 0;
58: }
59:
60: void *thread_function(void *arguments) {
61:     struct thread_arguments *p = (struct thread_arguments *)arguments;
62:     int accept_sd = p->accept_sd;
63:     free(p); // Free allocated memory
64:
65:     char buffer[80];
66:     time_t current_time;
67:     struct tm *time_info;
68:
69:     for (int i = 0; i < RUN_TIME / INTERVAL; i++) {  // Loop for 30 seconds
70:         current_time = time(NULL);
71:         time_info = localtime(&current_time);
72:         strftime(buffer, sizeof(buffer), "Current time: %H:%M:%S\n", time_info);
73:
74:         send(accept_sd, buffer, strlen(buffer), 0);
75:         sleep(INTERVAL);  // Wait 3 seconds before next update
76:     }
77:
78:     // Close client socket after 30 seconds
79:     printf("Client disconnected\n");
80:     close(accept_sd);
81:     pthread_exit(NULL);
82: }
```