**Table of Contents**

[Output 2](#_Toc193740764)

[Testing with all clients running 2](#_Toc193740765)

[Testing with Isolating one client 2](#_Toc193740766)

[Graceful Exiting 3](#_Toc193740767)

[Source Code : server.cpp 3](#_Toc193740768)

[Explanation of server 7](#_Toc193740769)

[Header Inclusions 7](#_Toc193740770)

[Global Variables 7](#_Toc193740771)

[Shutdown Handler 8](#_Toc193740772)

[Signal Handling Setup 8](#_Toc193740773)

[Message Queue Initialization 8](#_Toc193740774)

[Mutex Initialization 9](#_Toc193740775)

[Thread Creation (Message Receiving) 9](#_Toc193740776)

[Main Loop (Message Forwarding) 10](#_Toc193740777)

[Graceful Shutdown (Sending Quit Messages) 10](#_Toc193740778)

[Thread Cleanup & Message Queue Removal 11](#_Toc193740779)

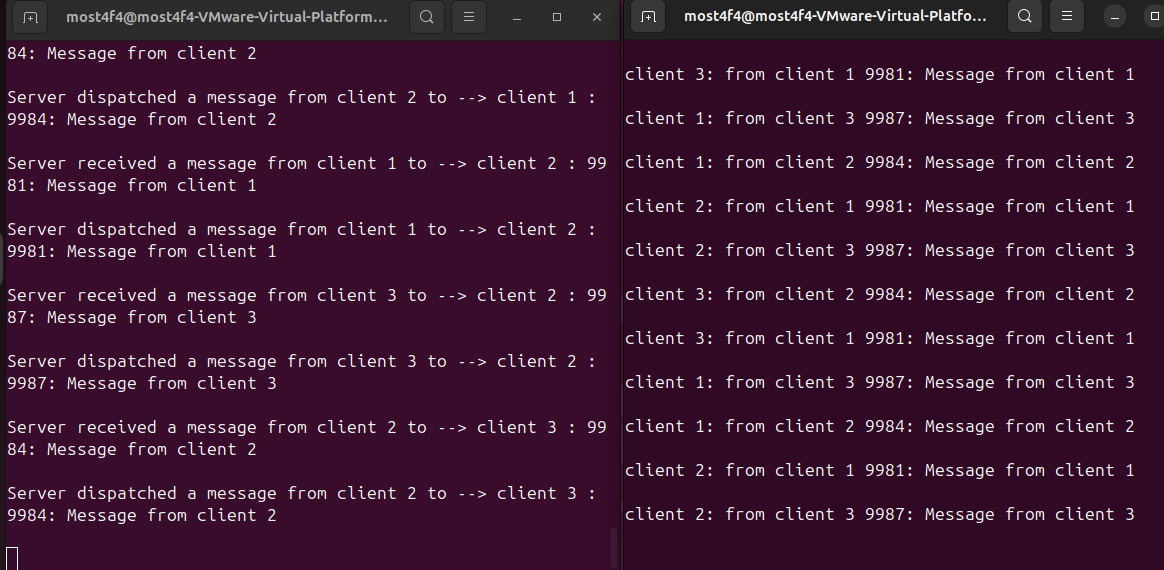
[Receiving Thread (recv\_func()) 11](#_Toc193740780)

[Summary 12](#_Toc193740781)

# 

# Output

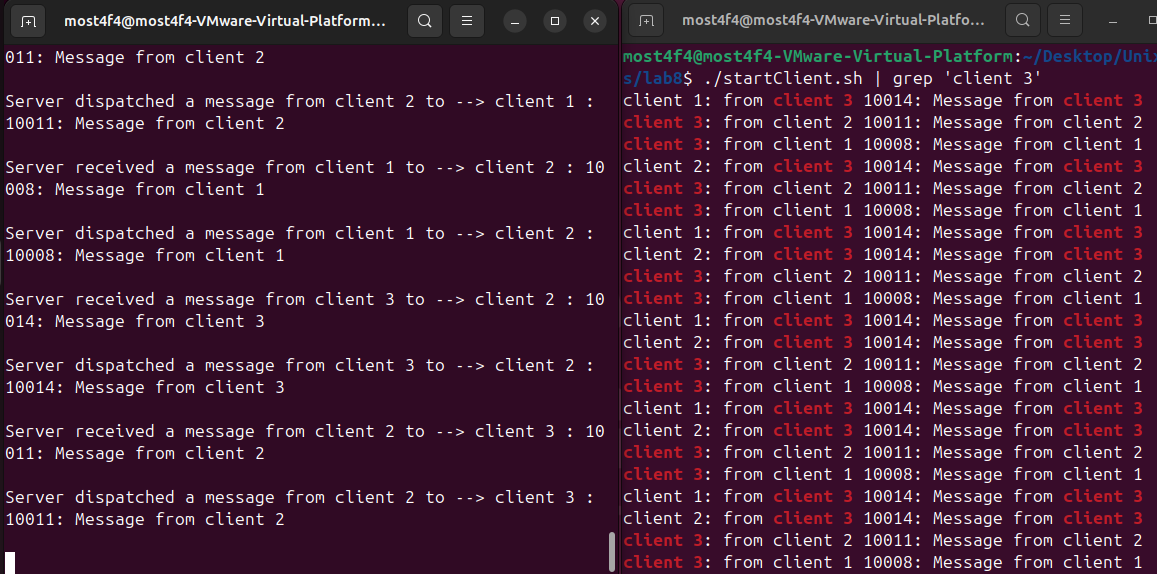
#### Testing with all clients running



#### 

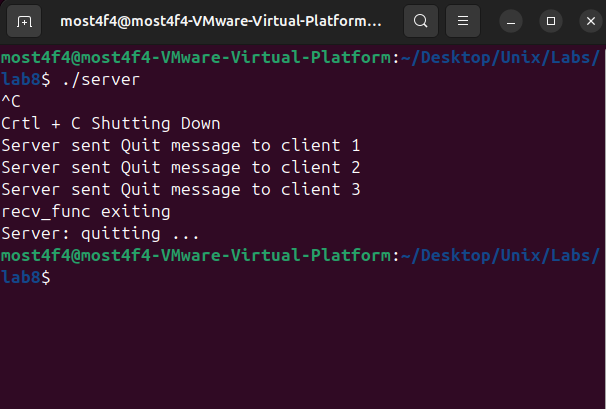
#### 

#### Testing with Isolating one client



#### 

#### Graceful Exiting



# Source Code : server.cpp

|  |
| --- |
| #include <errno.h>  #include <iostream>  #include <queue>  #include <signal.h>  #include <string.h> #include <sys/ipc.h>  #include <sys/msg.h>  #include <unistd.h> #include "client.h"  using namespace std;  key\_t key; int msgid; bool is\_running; queue<Message> message;  */\* shared mutex between receive thread and send \*/* pthread\_mutex\_t lock\_x;  void\* recv\_func(void\* arg);  static void shutdownHandler(int sig) {  switch (sig) {  case SIGINT:  cout << "\nCrtl + C Shutting Down" << endl;  is\_running = false;  break;  } }  int main() {  pthread\_t tid\_r;   *//Configue and set signal handler for SIGINT*  struct sigaction action;  action.sa\_handler = shutdownHandler;  sigemptyset(&action.sa\_mask);  action.sa\_flags = 0;  sigaction(SIGINT, &action, NULL);   *// Generate unique key for message queue*  key = ftok("serverclient", 65);  if (key == -1) {  cout << "Error generating key: " << strerror(errno) << endl;  return -1;  }   *// Create the message queue*  msgid = msgget(key, 0666 | IPC\_CREAT);  if (msgid == -1) {  cout << "Error creating message queue: " << strerror(errno) << endl;  return -1;  }   *// Initializes a mutex (lock)*  if (pthread\_mutex\_init(&lock\_x, NULL) != 0) {  cout << "Error initializing mutex: " << strerror(errno) << endl;  return -1;  }   is\_running = true;   *// Creates a new thread that runs the function recv\_func()*  if (pthread\_create(&tid\_r, NULL, recv\_func, NULL) != 0) {  is\_running = false;  cout << "Error creating receive thread: " << strerror(errno) << endl;  return -1;  }   while (is\_running) {  if (message.size() > 0) {  pthread\_mutex\_lock(&lock\_x);  Message sendMsg = message.front();  message.pop();  pthread\_mutex\_unlock(&lock\_x);   sendMsg.mtype = sendMsg.msgBuf.dest;  if (msgsnd(msgid, &sendMsg, sizeof(sendMsg), 0) == -1) {  cout << "Error sending message: " << strerror(errno) << endl;  }  else {  cout << "Server dispatched a message from client "  << sendMsg.msgBuf.source << " to --> client "  << sendMsg.msgBuf.dest << " : " << sendMsg.msgBuf.buf << endl;  }  }  else {  usleep(1000); *// Sleep briefly to avoid busy-waiting*  }  }   *// Send "Quit" messages to all clients on shutdown*  Message quitMsg;  sprintf(quitMsg.msgBuf.buf, "Quit");   for (int i = 0; i < 3; i++) {  quitMsg.mtype = i + 1;  quitMsg.msgBuf.source = 0;  quitMsg.msgBuf.dest = i + 1;  if (msgsnd(msgid, &quitMsg, sizeof(quitMsg), 0) == -1) {  cout << "Error sending Quit to client " << i + 1 << ": " << strerror(errno) << endl;  }  else {  cout << "Server sent Quit message to client " << i + 1 << endl;  }   }   *// Wait for receive thread to exit*  if (pthread\_join(tid\_r, NULL) != 0) {  cout << "Error joining receive thread: " << strerror(errno) << endl;  return -1;  }   cout << "Server: quitting ..." << endl;  msgctl(msgid, IPC\_RMID, NULL);  return 0; }  void\* recv\_func(void\* arg) {  while (is\_running) {  Message msg;  *// Use IPC\_NOWAIT with msgrcv to make it non-blocking, allowing the loop to check is\_running regularly.*  if (msgrcv(msgid, &msg, sizeof(msg), 4, IPC\_NOWAIT) == -1) {  if (errno == ENOMSG) { *// No message available*  usleep(1000); *// Brief sleep to avoid busy-waiting*  continue;  }  if (errno == EINTR || errno == EIDRM) { *// Signal or queue removed*  break;  }  cout << "Error receiving message: " << strerror(errno) << endl;  continue;  }  pthread\_mutex\_lock(&lock\_x);  message.push(msg);  cout << "Server received a message from client " << msg.msgBuf.source  << " to --> client " << msg.msgBuf.dest << " : " << msg.msgBuf.buf << endl;  pthread\_mutex\_unlock(&lock\_x);  }  cout << "recv\_func exiting" << endl; *// Debug to confirm exit*  pthread\_exit(NULL); } |

# Explanation of server

This code implements a simple message-passing server using **POSIX threads (pthread)** and **System V message queues**. It listens for messages from clients and forwards them to their intended recipients. It also handles graceful shutdown when receiving a SIGINT (Ctrl+C).

#### Header Inclusions

* errno.h → Provides error codes for system calls.
* iostream → For standard input/output.
* queue → Implements a **FIFO** message queue to store received messages.
* signal.h → Allows handling of signals (e.g., SIGINT).  
  string.h → Provides string manipulation functions (strerror is used for error messages).  
  sys/ipc.h → Used for **interprocess communication (IPC)**.
* sys/msg.h → Provides System V message queue functionality.
* unistd.h → Provides **usleep()**, which is used to pause execution.
* "client.h" → Presumably contains the **Message structure** and possibly client-side functions.

#### Global Variables

|  |
| --- |
| key\_t key; int msgid; bool is\_running; queue<Message> message; pthread\_mutex\_t lock\_x; |

* key\_t key; → Holds the unique **IPC key** for the message queue.
* int msgid; → Stores the **message queue identifier**.
* bool is\_running; → Indicates whether the server is **actively running**.
* queue<Message> message; → A **FIFO queue** to temporarily store received messages.
* pthread\_mutex\_t lock\_x; → A **mutex (lock)** to synchronize access to the message queue.

#### Shutdown Handler

|  |
| --- |
| static void shutdownHandler(int sig) {  switch (sig) {  case SIGINT:  cout << "\nCrtl + C Shutting Down" << endl;  is\_running = false;  break;  } } |

* This function is triggered when SIGINT (**Ctrl+C**) is received.
* It simply sets is\_running = false;, signaling the server to shut down.
* The use of static ensures this function is **local to the file**.

#### Signal Handling Setup

|  |
| --- |
| struct sigaction action; action.sa\_handler = shutdownHandler; sigemptyset(&action.sa\_mask); action.sa\_flags = 0; sigaction(SIGINT, &action, NULL); |

* Registers shutdownHandler as the handler for **SIGINT**.
* Ensures that the process **does not terminate immediately** on Ctrl+C, but instead **gracefully exits**.

#### Message Queue Initialization

|  |
| --- |
| key = ftok("serverclient", 65); if (key == -1) {  cout << "Error generating key: " << strerror(errno) << endl;  return -1; } |

* ftok("serverclient", 65); generates a **unique IPC key** based on a file ("serverclient") and an integer (65).
* If key generation fails, the error is printed.

|  |
| --- |
| msgid = msgget(key, 0666 | IPC\_CREAT); if (msgid == -1) {  cout << "Error creating message queue: " << strerror(errno) << endl;  return -1; } |

* msgget() **creates** or **retrieves** a **message queue** with **read/write permissions (0666)**.
* If it fails, an error message is displayed.

#### Mutex Initialization

|  |
| --- |
| if (pthread\_mutex\_init(&lock\_x, NULL) != 0) {  cout << "Error initializing mutex: " << strerror(errno) << endl;  return -1; } |

* Initializes lock\_x, a **mutex for synchronizing message access**.
* If initialization fails, the program exits.

#### Thread Creation (Message Receiving)

|  |
| --- |
| is\_running = true;  if (pthread\_create(&tid\_r, NULL, recv\_func, NULL) != 0) {  is\_running = false;  cout << "Error creating receive thread: " << strerror(errno) << endl;  return -1; } |

* is\_running = true; ensures the server remains active.
* pthread\_create() spawns a **separate thread** (tid\_r) to run recv\_func().
* If thread creation fails, an error is printed, and is\_running is set to **false**.

#### Main Loop (Message Forwarding)

|  |
| --- |
| while (is\_running) {  if (message.size() > 0) {  pthread\_mutex\_lock(&lock\_x);  Message sendMsg = message.front();  message.pop();  pthread\_mutex\_unlock(&lock\_x); |

* This **main loop** runs **while the server is active**.
* If there are messages in the queue:
  + **Mutex lock** is acquired (pthread\_mutex\_lock(&lock\_x);).
  + The **first message** is retrieved and **removed** from the queue (message.pop();).
  + The mutex is then **released** (pthread\_mutex\_unlock(&lock\_x);).

|  |
| --- |
| sendMsg.mtype = sendMsg.msgBuf.dest;  if (msgsnd(msgid, &sendMsg, sizeof(sendMsg), 0) == -1) {  cout << "Error sending message: " << strerror(errno) << endl;  } else {  cout << "Server dispatched a message from client "  << sendMsg.msgBuf.source << " to --> client "  << sendMsg.msgBuf.dest << " : " << sendMsg.msgBuf.buf << endl;  } } else {  usleep(1000); *// Sleep briefly to avoid busy-waiting* } |

* The **message type** (mtype) is set to the **destination client ID**.
* The message is **sent** via msgsnd().
* If the queue is empty, usleep(1000); prevents **busy-waiting**.

#### Graceful Shutdown (Sending Quit Messages)

|  |
| --- |
| Message quitMsg; sprintf(quitMsg.msgBuf.buf, "Quit");  for (int i = 0; i < 3; i++) {  quitMsg.mtype = i + 1;  quitMsg.msgBuf.source = 0;  quitMsg.msgBuf.dest = i + 1;  if (msgsnd(msgid, &quitMsg, sizeof(quitMsg), 0) == -1) {  cout << "Error sending Quit to client " << i + 1 << ": " << strerror(errno) << endl;  }  else {  cout << "Server sent Quit message to client " << i + 1 << endl;  } } |

* A **"Quit" message** is created.
* The server sends it to **clients 1, 2, and 3** before shutting down.

#### Thread Cleanup & Message Queue Removal

|  |
| --- |
| if (pthread\_join(tid\_r, NULL) != 0) {  cout << "Error joining receive thread: " << strerror(errno) << endl;  return -1; } |

* Ensures the **receive thread exits cleanly**.

|  |
| --- |
| cout << "Server: quitting ..." << endl; msgctl(msgid, IPC\_RMID, NULL); return 0; |

* **Deletes** the message queue using msgctl(msgid, IPC\_RMID, NULL);.

#### Receiving Thread (recv\_func())

|  |
| --- |
| void\* recv\_func(void\* arg) {  while (is\_running) {  Message msg;  if (msgrcv(msgid, &msg, sizeof(msg), 4, IPC\_NOWAIT) == -1) {  if (errno == ENOMSG) {  usleep(1000);  continue;  }  if (errno == EINTR || errno == EIDRM) {  break;  }  cout << "Error receiving message: " << strerror(errno) << endl;  continue;  } |

* Runs in a **separate thread**.
* Uses msgrcv() to **retrieve messages of type 4** (possibly broadcast messages).
* If **no message is available**, it **sleeps briefly**.
* If EIDRM (queue removed) or EINTR (interrupted), it **exits**.

|  |
| --- |
| pthread\_mutex\_lock(&lock\_x);  message.push(msg);  cout << "Server received a message from client " << msg.msgBuf.source  << " to --> client " << msg.msgBuf.dest << " : " << msg.msgBuf.buf << endl;  pthread\_mutex\_unlock(&lock\_x); } cout << "recv\_func exiting" << endl; pthread\_exit(NULL); |

* Stores the message in the **queue**, protected by **mutex**.

#### Summary

This **multithreaded** server:

* **Receives messages** from clients (recv\_func).
* **Forwards messages** to the intended recipients.
* **Gracefully shuts down** when Ctrl+C is pressed.
* Uses **mutex locking** to ensure thread safety.