**Project-Multi-Process To-Do List Application**

**(using shared memory)**

**Introduction**

This project is a simple client-server application that demonstrates the use of Inter-Process Communication (IPC) mechanisms in C++, specifically using shared memory and semaphores. The project simulates a to-do list management system where multiple processes can concurrently add, complete, and view tasks in a shared environment. The implementation showcases key concepts in systems programming, such as process synchronization and shared memory management.

**Project Objective**

The primary objective of this project is to create a multi-process application that effectively uses IPC to manage a shared resource—in this case, a to-do list. The goals are:

1. To demonstrate the use of shared memory for communication between processes.

2. To implement semaphore-based synchronization to prevent race conditions.

3. To create a simple yet functional to-do list application that allows users to add tasks, mark them as completed, and view the task list.

**System Design**

1. **Shared Memory**: The shared memory segment is used to store the to-do list, which includes an array of To-Do Item structures. Each To-Do Item contains a description and a completion status.

2. **Semaphores**: A semaphore is used to synchronize access to the shared memory. This ensures that only one process can modify the to-do list at a time, preventing data corruption due to concurrent writes.

3. **Processes**: The application runs in a loop that accepts commands from the user (e.g., add, complete, view, exit). Each command triggers an operation that either updates the shared memory or reads from it, all while respecting the semaphore locks.

4**. User Interaction**: Users can interact with the system via a command-line interface. They can add new to-do items, mark existing items as completed, and view the current list of tasks.

**Source code**

**Server.cpp**

#include <iostream>

#include <string>

#include <sys/shm.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <semaphore.h>

#include <unistd.h>

#include <cstring>

// Structure to represent a to-do item

struct ToDoItem {

char description[256];

bool completed;

};

// Shared memory segment structure

struct SharedMemory {

int numItems;

ToDoItem items[10];

// Fixed-size array for simplicity

sem\_t sem;

// Semaphore for synchronization

};

// Function to add a new to-do item to the shared memory segment

void addToDoItem(SharedMemory\* shmPtr, const char\* description) {

sem\_wait(&shmPtr->sem);

// Acquire the semaphore

strcpy(shmPtr->items[shmPtr->numItems].description, description);

shmPtr->items[shmPtr->numItems].completed = false;

shmPtr->numItems++;

sem\_post(&shmPtr->sem);

// Release the semaphore

}

// Function to complete a to-do item

void completeToDoItem(SharedMemory\* shmPtr, int index) {

sem\_wait(&shmPtr->sem); // Acquire the semaphore

if (index < shmPtr->numItems) {

shmPtr->items[index].completed = true;

}

sem\_post(&shmPtr->sem);

// Release the semaphore

}

// Function to print the to-do list

void printToDoList(SharedMemory\* shmPtr) {

sem\_wait(&shmPtr->sem); // Acquire the semaphore

std::cout << "To-Do List:" << std::endl;

for (int i = 0; i < shmPtr->numItems; i++) {

std::cout << i + 1 << ". " << shmPtr->items[i].description

<< " [" << (shmPtr->items[i].completed ? "Completed" : "Pending") << "]" << std::endl;

}

sem\_post(&shmPtr->sem);

// Release the semaphore

}

int main() {

// Create shared memory segment

int shmId = shmget(IPC\_PRIVATE, sizeof(SharedMemory), IPC\_CREAT | 0666);

if (shmId == -1) {

perror("shmget");

return 1;

}

// Attach to the shared memory segment

SharedMemory\* shmPtr = (SharedMemory\*) shmat(shmId, NULL, 0);

if (shmPtr == (void\*) -1) {

perror("shmat");

return 1;

}

// Initialize shared memory

shmPtr->numItems = 0;

sem\_init(&shmPtr->sem, 1, 1);

// Initialize semaphore for shared memory

// Add some to-do items

addToDoItem(shmPtr, "Design webpage");

addToDoItem(shmPtr, "Do backend");

addToDoItem(shmPtr, "Deploy website");

// Print the to-do list

printToDoList(shmPtr);

// Wait to keep the server running

std::cout << "Server running. Press Enter to exit..." << std::endl;

std::cin.get();

// Clean up

sem\_destroy(&shmPtr->sem);

shmdt(shmPtr);

shmctl(shmId, IPC\_RMID, 0);

return 0;

}

**Client.cpp**

#include <iostream>

#include <string>

#include <sys/shm.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <semaphore.h>

#include <unistd.h>

#include <cstring>

// Structure to represent a to-do item

struct ToDoItem {

char description[256];

bool completed;

};

// Shared memory segment structure

struct SharedMemory {

int numItems;

ToDoItem items[10];

// Fixed-size array for simplicity

sem\_t sem;

// Semaphore for synchronization

};

// Function to add a new to-do item to the shared memory segment

void addToDoItem(SharedMemory\* shmPtr, const char\* description) {

sem\_wait(&shmPtr->sem);

// Acquire the semaphore

strcpy(shmPtr->items[shmPtr->numItems].description, description);

shmPtr->items[shmPtr->numItems].completed = false;

shmPtr->numItems++;

sem\_post(&shmPtr->sem);

// Release the semaphore

}

// Function to complete a to-do item

void completeToDoItem(SharedMemory\* shmPtr, int index) {

sem\_wait(&shmPtr->sem);

// Acquire the semaphore

if (index < shmPtr->numItems) {

shmPtr->items[index].completed = true;

}

sem\_post(&shmPtr->sem);

// Release the semaphore

}

// Function to print the to-do list

void printToDoList(SharedMemory\* shmPtr) {

sem\_wait(&shmPtr->sem);

// Acquire the semaphore

std::cout << "To-Do List:" << std::endl;

for (int i = 0; i < shmPtr->numItems; i++) {

std::cout << i + 1 << ". " << shmPtr->items[i].description

<< " [" << (shmPtr->items[i].completed ? "Completed" : "Pending") << "]" << std::endl;

}

sem\_post(&shmPtr->sem);

// Release the semaphore

}

int main() {

// Create shared memory segment

int shmId = shmget(IPC\_PRIVATE, sizeof(SharedMemory), IPC\_CREAT | 0666);

if (shmId == -1) {

perror("shmget");

return 1;

}

// Attach to the shared memory segment

SharedMemory\* shmPtr = (SharedMemory\*) shmat(shmId, NULL, 0);

if (shmPtr == (void\*) -1) {

perror("shmat");

return 1;

}

// Initialize shared memory

shmPtr->numItems = 0;

sem\_init(&shmPtr->sem, 1, 1);

// Initialize semaphore for shared memory

// User interaction loop

while (true) {

std::string command;

std::cout << "Enter command (add, complete, view, exit): ";

std::cin >> command;

if (command == "add") {

std::string description;

std::cout << "Enter description: ";

std::cin.ignore();

std::getline(std::cin, description);

addToDoItem(shmPtr, description.c\_str());

} else if (command == "complete") {

int index;

std::cout << "Enter item number to complete: ";

std::cin >> index;

completeToDoItem(shmPtr, index - 1);

} else if (command == "view") {

printToDoList(shmPtr);

} else if (command == "exit") {

break;

} else {

std::cout << "Invalid command." << std::endl;

}

}

// Clean up

sem\_destroy(&shmPtr->sem);

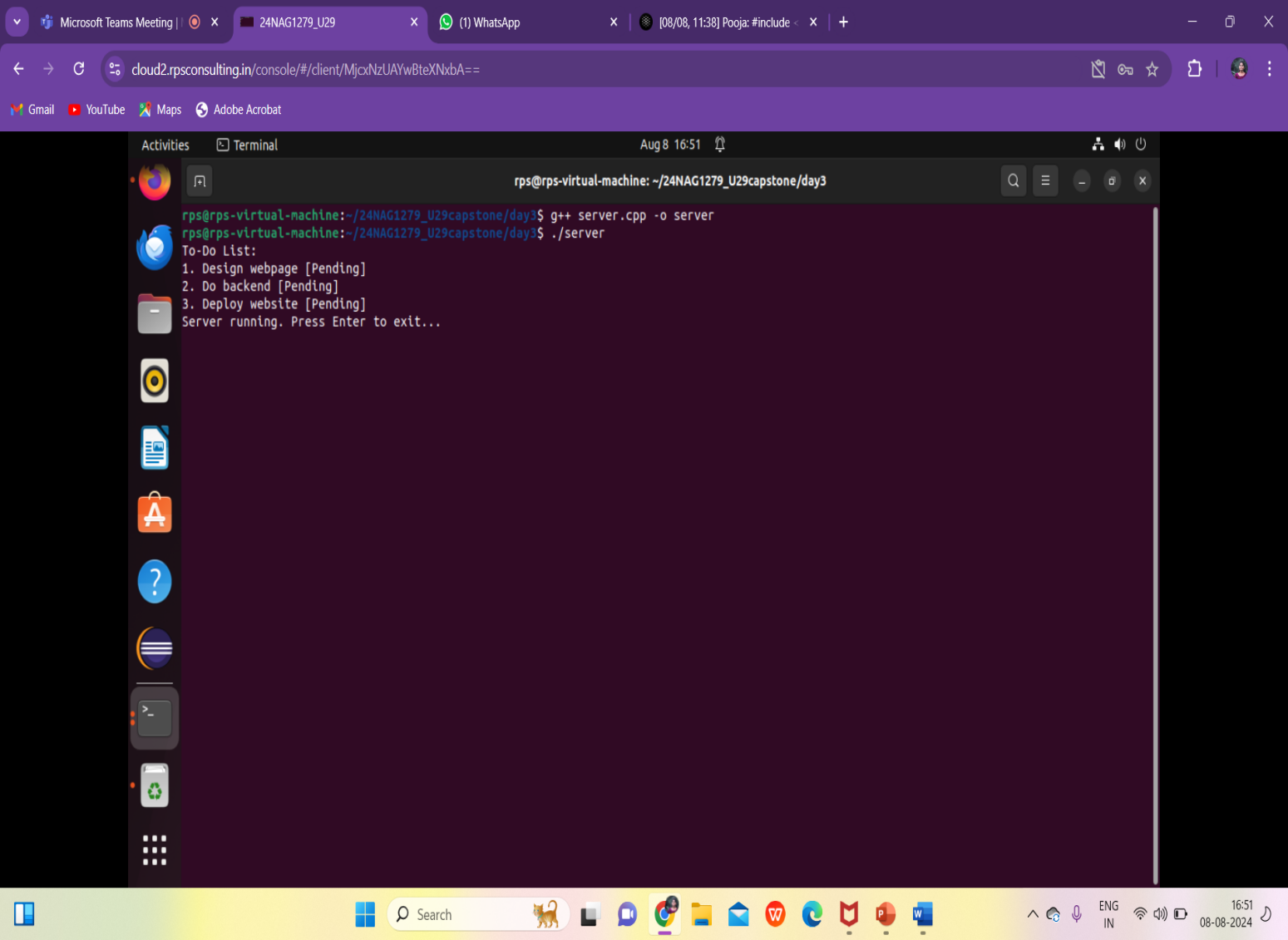
shmdt(shmPtr);

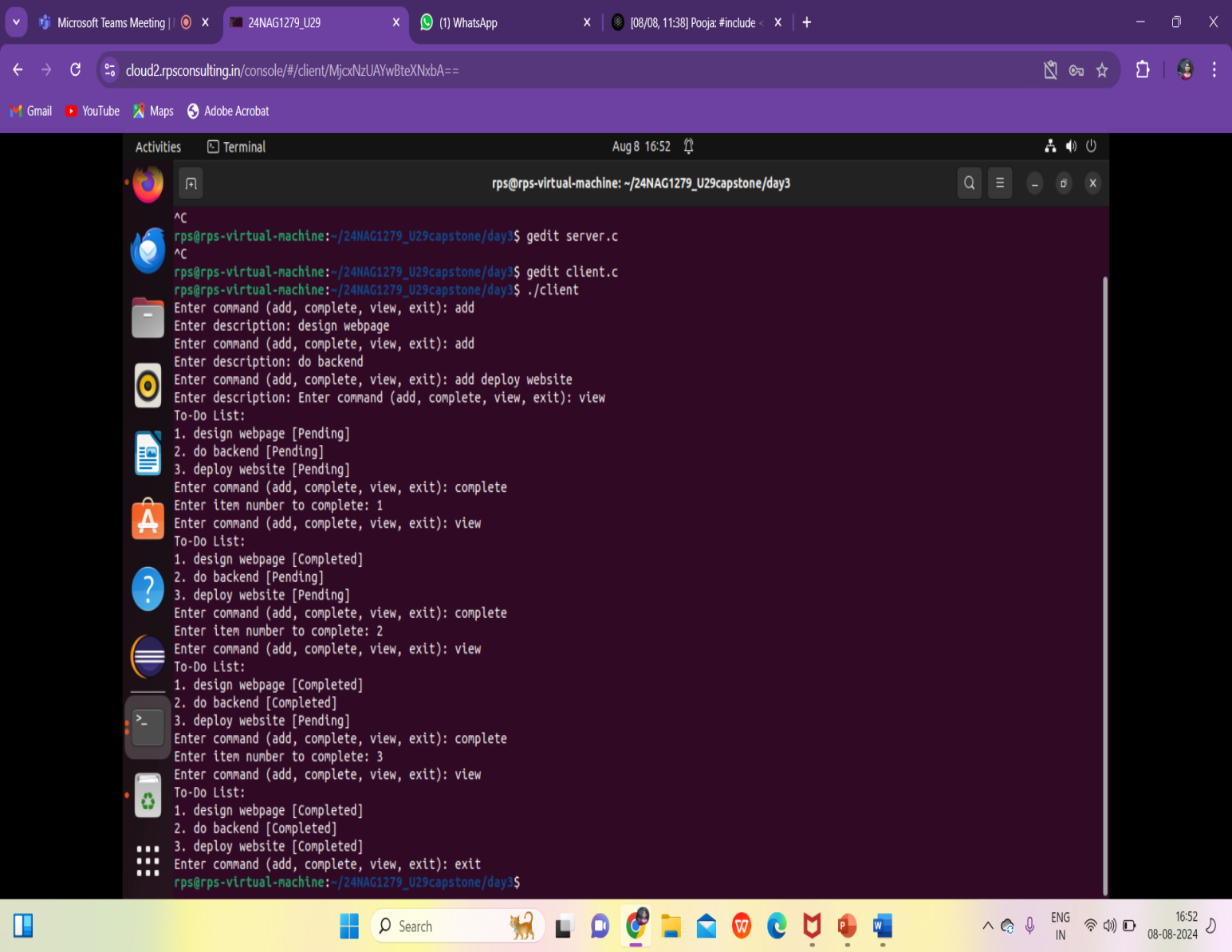
shmctl(shmId, IPC\_RMID, 0);

return 0;

}

**Output**





**Conclusion**

The project successfully demonstrates the practical application of IPC using shared memory and semaphores in a multi-process environment. Through the creation of a simple to-do list application, it highlights the importance of synchronization in preventing race conditions and ensuring data integrity. The system is designed to be simple yet functional, providing a strong foundation for further enhancements that could add more features and improve usability. The project serves as a valuable learning tool for understanding the complexities of process communication and resource sharing in systems programming.