# POSIX IPC: Semaphores, Shared Memory, and Message Queues (Detailed Explanation)

## 1. POSIX Semaphores

Semaphores are synchronization mechanisms used to control access to shared resources. POSIX provides both named (`sem\_open`) and unnamed (`sem\_init`) semaphores.

### 1.1 Unnamed Semaphores (Thread Synchronization)

Unnamed semaphores are used within a single process for synchronizing threads.

#include <iostream> // For standard input-output operations  
#include <pthread.h> // For POSIX threads  
#include <semaphore.h> // For semaphore functions  
#include <unistd.h> // For sleep function  
  
sem\_t sem; // Declare a semaphore  
  
void\* thread\_function(void\* arg) {  
 sem\_wait(&sem); // Decrease semaphore value, block if it is zero  
 std::cout << "Thread " << \*(int\*)arg << " entered critical section  
";  
 sleep(1); // Simulate some processing time  
 std::cout << "Thread " << \*(int\*)arg << " exiting critical section  
";  
 sem\_post(&sem); // Increase semaphore value, allowing another thread to enter  
 return nullptr;  
}  
  
int main() {  
 pthread\_t threads[3]; // Declare an array of 3 threads  
 int ids[3] = {1, 2, 3}; // Thread IDs  
  
 sem\_init(&sem, 0, 1); // Initialize semaphore: 0 = shared within process, 1 = initial value  
  
 for (int i = 0; i < 3; i++)  
 pthread\_create(&threads[i], nullptr, thread\_function, &ids[i]); // Create threads  
  
 for (int i = 0; i < 3; i++)  
 pthread\_join(threads[i], nullptr); // Wait for all threads to finish  
  
 sem\_destroy(&sem); // Destroy the semaphore  
 return 0;  
}

### 1.2 Named Semaphores (Process Synchronization)

#include <iostream> // For standard I/O  
#include <semaphore.h> // For semaphore functions  
#include <fcntl.h> // For O\_CREAT flag  
#include <unistd.h> // For sleep  
  
#define SEM\_NAME "/my\_semaphore" // Define a unique name for the semaphore  
  
int main() {  
 sem\_t \*sem = sem\_open(SEM\_NAME, O\_CREAT, 0644, 1);   
 // sem\_open creates or opens a named semaphore  
 // Arguments:   
 // SEM\_NAME: Unique name for semaphore  
 // O\_CREAT: Creates the semaphore if it does not exist  
 // 0644: File permissions  
 // 1: Initial semaphore value

// can be 3 if allow 3 process at a time ie block 4th

//for binary semaphore give 1 since allow only 1  
  
 sem\_wait(sem); // Lock the semaphore (decrease value)  
 std::cout << "Semaphore Acquired  
";  
 sem\_post(sem); // Release the semaphore (increase value)  
 sem\_close(sem); // Close the semaphore  
 sem\_unlink(SEM\_NAME); // Remove the named semaphore  
 return 0;  
}

## 2. Shared Memory with Semaphores

#include <iostream> // Standard I/O  
#include <fcntl.h> // File control options  
#include <sys/mman.h> // Memory management  
#include <sys/stat.h> // File permissions  
#include <semaphore.h> // Semaphores  
#include <unistd.h> // Sleep function  
  
#define SHM\_NAME "/shm\_example" // Shared memory name  
#define SEM\_MUTEX "/sem\_mutex" // Semaphore name  
  
struct SharedMemory { int data; }; // Define shared memory structure  
  
int main() {  
 int shm\_fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);  
 // shm\_open creates shared memory object  
 // O\_CREAT | O\_RDWR: Create and open in read/write mode  
 // 0666: File permissions  
  
 ftruncate(shm\_fd, sizeof(SharedMemory));   
 // Resize shared memory to fit structure  
  
 SharedMemory\* shm = (SharedMemory\*)mmap(0, sizeof(SharedMemory), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);  
 // mmap maps shared memory to process address space  
  
 sem\_t\* mutex = sem\_open(SEM\_MUTEX, O\_CREAT, 0666, 1); // Create semaphore  
  
 for (int i = 1; i <= 5; i++) {  
 sem\_wait(mutex); // Lock semaphore  
 shm->data = i; // Write data to shared memory  
 std::cout << "Produced: " << i << std::endl;  
 sem\_post(mutex); // Release semaphore  
 sleep(1);  
 }  
  
 sem\_close(mutex);  
 munmap(shm, sizeof(SharedMemory)); // Unmap shared memory  
 close(shm\_fd); // Close shared memory descriptor  
 return 0;  
}

## 3. POSIX Message Queues

#include <iostream> // Standard I/O  
#include <mqueue.h> // Message Queue APIs  
#include <cstring> // String manipulation  
  
#define QUEUE\_NAME "/my\_queue" // Unique message queue name  
  
int main() {  
 mqd\_t mq = mq\_open(QUEUE\_NAME, O\_CREAT | O\_WRONLY, 0644, nullptr);  
 // mq\_open creates or opens a message queue  
 // O\_CREAT | O\_WRONLY: Create and open in write mode  
 // 0644: Permissions  
  
 const char\* message = "Hello, Message Queue!";  
 mq\_send(mq, message, strlen(message) + 1, 0); // Send message to queue  
  
 mq\_close(mq); // Close message queue  
 return 0;  
}

### 3.2 Message Queue Receiver

#include <iostream> // Standard I/O  
#include <mqueue.h> // Message queue functions  
  
#define QUEUE\_NAME "/my\_queue" // Same queue name as sender  
  
int main() {  
 mqd\_t mq = mq\_open(QUEUE\_NAME, O\_RDONLY); // Open queue in read mode  
 char buffer[256]; // Buffer for receiving message  
  
 mq\_receive(mq, buffer, 256, nullptr); // Receive message from queue  
 std::cout << "Received: " << buffer << std::endl;  
  
 mq\_close(mq); // Close message queue  
 return 0;  
}