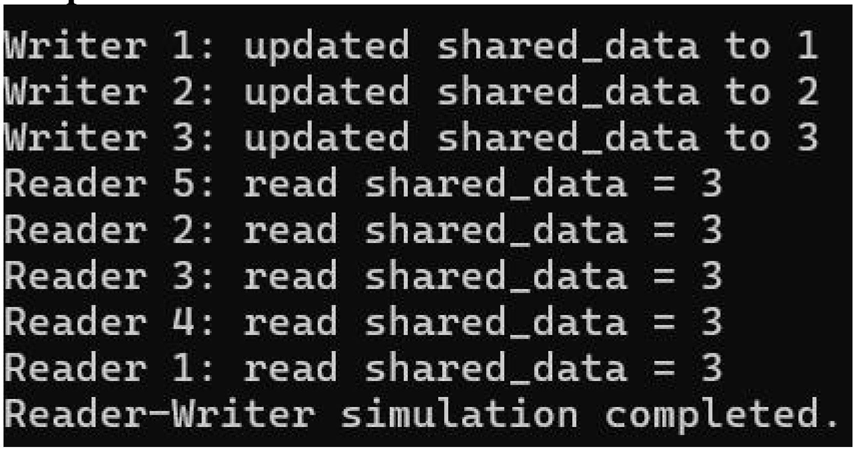
18. Construct a C program to simulate Reader-Writer problem using Semaphores.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

sem_t rw_mutex; // Semaphore to control access to the shared
resource
sem_t mutex;    // Semaphore to ensure mutual exclusion for the read
count
int read_count = 0; // Count of active readers
int shared_data = 0; // Shared resource

void* reader(void* arg) {
   int id = *(int*)arg;

   sem_wait(&mutex); // Lock for updating read_count
   read_count++;
   if (read_count == 1) {
      sem_wait(&rw_mutex); // First reader locks the shared resource
   }
   sem_post(&mutex); // Unlock after updating read_count

   // Reading shared resource
   printf("Reader %d: read shared_data = %d\n", id, shared_data);
   sleep(1);

   sem_wait(&mutex); // Lock for updating read_count
   read_count--;
   if (read_count == 0) {
      sem_post(&rw_mutex); // Last reader unlocks the shared
resource
   }
   sem_post(&mutex); // Unlock after updating read_count

   free(arg);
   return NULL;
}

void* writer(void* arg) {
   int id = *(int*)arg;

   sem_wait(&rw_mutex); // Writer locks the shared resource
   shared_data += 1;    // Writing to the shared resource
   printf("Writer %d: updated shared_data to %d\n", id,
shared_data);
   sleep(1);
   sem_post(&rw_mutex); // Unlock the shared resource

   free(arg);
   return NULL;
}

int main() {
   int num_readers = 5, num_writers = 3;
   pthread_t readers[num_readers], writers[num_writers];

   // Initialize semaphores
   sem_init(&rw_mutex, 0, 1);
   sem_init(&mutex, 0, 1);

   // Create writer threads
   for (int i = 0; i < num_writers; i++) {
      int* id = malloc(sizeof(int));
      *id = i + 1;
      pthread_create(&writers[i], NULL, writer, id);
   }

   // Create reader threads
   for (int i = 0; i < num_readers; i++) {
      int* id = malloc(sizeof(int));
      *id = i + 1;
      pthread_create(&readers[i], NULL, reader, id);
   }

   // Wait for all writers to finish
   for (int i = 0; i < num_writers; i++) {
      pthread_join(writers[i], NULL);
   }

   // Wait for all readers to finish
   for (int i = 0; i < num_readers; i++) {
      pthread_join(readers[i], NULL);
   }

   // Destroy semaphores
   sem_destroy(&rw_mutex);
   sem_destroy(&mutex);

   printf("Reader-Writer simulation completed.\n");
   return 0;
}
```

Output:



```
Writer 1: updated shared_data to 1
Writer 2: updated shared_data to 2
Writer 3: updated shared_data to 3
Reader 5: read shared_data = 3
Reader 2: read shared_data = 3
Reader 3: read shared_data = 3
Reader 4: read shared_data = 3
Reader 1: read shared_data = 3
Reader-Writer simulation completed.
```