**Q1.**

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <time.h>

#define BUFFER\_SIZE 10

int buffer[BUFFER\_SIZE]; // Shared buffer

int count = 0; // Number of items in the buffer

// Semaphores for synchronization

sem\_t mutex, full, empty;

void \*producer(void \*arg) {

int i, item;

for (i = 0; i < 20; i++) {

item = rand() % 100; // Produce a random item

printf("Producer is waiting\n");

sem\_wait(&empty); // Wait for an empty slot

sem\_wait(&mutex); // Enter critical section (mutex)

printf("Producer acquires Mutex\n");

// Produce item

buffer[count++] = item;

printf("Producer produces item: [%d]\n", item);

sem\_post(&mutex); // Exit critical section (mutex)

printf(" Producer releases Mutex......\n");

sem\_post(&full); // Signal that a new item is produced

sleep(1); // Simulate production time

}

pthread\_exit(NULL); // Exit the thread

}

void \*consumer(void \*arg) {

int i, item;

for (i = 0; i < 20; i++) {

printf("Consumer is waiting\n");

sem\_wait(&full); // Wait for an item to consume

sem\_wait(&mutex); // Enter critical section (mutex)

printf("Consumer acquires Mutex\n");

// Consume item

item = buffer[--count];

printf("Consumer consumes item: [%d]\n", item);

sem\_post(&mutex); // Exit critical section (mutex)

printf(" Consumer releases Mutex......\n");

sem\_post(&empty); // Signal that a slot is free

sleep(1); // Simulate consumption time

}

pthread\_exit(NULL); // Exit the thread

}

int main() {

pthread\_t thread1, thread2;

// Initialize semaphores

sem\_init(&empty, 0, BUFFER\_SIZE); // Initially, the buffer is empty

sem\_init(&full, 0, 0); // Initially, there are no items in the buffer

sem\_init(&mutex, 0, 1); // Mutex is initially unlocked

srand(time(NULL)); // Seed the random number generator

// Create the producer and consumer threads

pthread\_create(&thread1, NULL, producer, NULL);

pthread\_create(&thread2, NULL, consumer, NULL);

// Wait for both threads to complete

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

// Clean up semaphores

sem\_destroy(&empty);

sem\_destroy(&full);

sem\_destroy(&mutex);

return 0;

}

**Output**

Producer is waiting

Producer acquires Mutex

Producer produces item: [89]

Producer releases Mutex......

Consumer is waiting

Consumer acquires Mutex

Consumer consumes item: [89]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [44]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [44]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [4]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [4]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [78]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [78]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [30]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [30]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [79]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [79]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [2]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [2]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [40]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [40]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [41]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [41]

Consumer releases Mutex......

Producer is waiting

Producer acquires Mutex

Producer produces item: [68]

Producer releases Mutex......

Consumer is waiting

Consumer acquires Mutex

Consumer consumes item: [68]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [16]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [16]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [51]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [51]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [46]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [46]

Consumer releases Mutex......

Producer is waiting

Producer acquires Mutex

Producer produces item: [59]

Producer releases Mutex......

Consumer is waiting

Consumer acquires Mutex

Consumer consumes item: [59]

Consumer releases Mutex......

Consumer is waiting

Producer is waiting

Producer acquires Mutex

Producer produces item: [10]

Producer releases Mutex......

Consumer acquires Mutex

Consumer consumes item: [10]

Consumer releases Mutex......

Producer is waiting

Producer acquires Mutex

Producer produces item: [99]

Producer releases Mutex......

Consumer is waiting

Consumer acquires Mutex

Consumer consumes item: [99]

Consumer releases Mutex......

Producer is waiting

Producer acquires Mutex

Producer produces item: [9]

Producer releases Mutex......

Consumer is waiting

Consumer acquires Mutex

Consumer consumes item: [9]

Consumer releases Mutex......

Producer is waiting

Producer acquires Mutex

Producer produces item: [41]

Producer releases Mutex......

Consumer is waiting

Consumer acquires Mutex

Consumer consumes item: [41]

Consumer releases Mutex......

Producer is waiting

Producer acquires Mutex

Producer produces item: [1]

Producer releases Mutex......

Consumer is waiting

Consumer acquires Mutex

Consumer consumes item: [1]

Consumer releases Mutex......

Producer is waiting

Producer acquires Mutex

Producer produces item: [72]

Producer releases Mutex......

Consumer is waiting

Consumer acquires Mutex

Consumer consumes item: [72]

Consumer releases Mutex......

**Q2.**

#include<stdio.h>

#include<unistd.h>

#include<pthread.h>

#include<semaphore.h>

#include<stdlib.h>

#define buffer\_size 10

int buffer[buffer\_size];

sem\_t mutex,s1,s2;

int count = 0;

void \*producer(void \*param)

{

int i, item;

for(i=0;i<=20;i++)

{

sem\_wait(&s2);

if(i%2!=0)

{

printf("Producer prints Odd number: %d\n", i);

sleep(1);

}

sem\_post(&s1);

}

pthread\_exit(NULL);

}

void \*consumer(void \*param)

{

int i, item;

for(i=0;i<=20;i++)

{

sem\_wait(&s1);

if(i%2==0)

{

printf("Consumer prints Even number: %d\n", i);

sleep(1);

}

sem\_post(&s2);

}

pthread\_exit(NULL);

}

int main()

{

pthread\_t thread1, thread2;

sem\_init(&s1,0,1);

sem\_init(&s2,0,0);

pthread\_create(&thread1,NULL,producer,NULL);

pthread\_create(&thread2,NULL,consumer,NULL);

pthread\_join(thread1,NULL);

pthread\_join(thread2,NULL);

return 0;

}

**Output**

srikant@SRIKANTs-PC:~/DOS\_2241016519/ass5$ ./q2

Consumer prints Even number: 0

Producer prints Odd number: 1

Consumer prints Even number: 2

Producer prints Odd number: 3

Consumer prints Even number: 4

Producer prints Odd number: 5

Consumer prints Even number: 6

Producer prints Odd number: 7

Consumer prints Even number: 8

Producer prints Odd number: 9

Consumer prints Even number: 10

Producer prints Odd number: 11

Consumer prints Even number: 12

Producer prints Odd number: 13

Consumer prints Even number: 14

Producer prints Odd number: 15

Consumer prints Even number: 16

Producer prints Odd number: 17

Consumer prints Even number: 18

Producer prints Odd number: 19

Consumer prints Even number: 20

**Q3.**

#include<stdio.h>

#include<unistd.h>

#include<pthread.h>

#include<semaphore.h>

#include<stdlib.h>

#define buffer\_size 10

int buffer[buffer\_size];

sem\_t mutex,s1,s2;

int count = 0;

void \*producer(void \*param)

{

int i, item;

for(i=1;i<=20;i++)

{

sem\_wait(&s1);

if(i%2!=0)

{

printf("%d. Thread A prints A\n",i);

sleep(1);

}

sem\_post(&s2);

}

pthread\_exit(NULL);

}

void \*consumer(void \*param)

{

int i, item;

for(i=1;i<=20;i++)

{

sem\_wait(&s2);

if(i%2==0)

{

printf("%d. Thread B prints B\n",i);

sleep(1);

}

sem\_post(&s1);

}

pthread\_exit(NULL);

}

int main()

{

pthread\_t thread1, thread2;

sem\_init(&s1,0,1); sem\_init(&s2,0,0);

pthread\_create(&thread1,NULL,producer,NULL);

pthread\_create(&thread2,NULL,consumer,NULL);

pthread\_join(thread1,NULL);

pthread\_join(thread2,NULL);

return 0;}

**Output:**

srikant@SRIKANTs-PC:~/DOS\_2241016519/ass5$ ./q3

1. Thread A prints A

2. Thread B prints B

3. Thread A prints A

4. Thread B prints B

5. Thread A prints A

6. Thread B prints B

7. Thread A prints A

8. Thread B prints B

9. Thread A prints A

10. Thread B prints B

11. Thread A prints A

12. Thread B prints B

13. Thread A prints A

14. Thread B prints B

15. Thread A prints A

16. Thread B prints B

17. Thread A prints A

18. Thread B prints B

19. Thread A prints A

20. Thread B prints B

**Q4.**

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#define BUFFER\_SIZE 10

// Semaphores for synchronization

sem\_t s1, s2;

int count = 10; // Shared variable to keep track of the count

// Producer function: counts down from 10 to 1

void \*producer(void \*param) {

int i;

for (i = 10; i > 0; i--) {

sem\_wait(&s1); // Wait for the consumer to finish

printf("Producer (Thread A) counts down: %d\n", i);

sleep(1); // Simulate work

sem\_post(&s2); // Signal the consumer to count up

}

pthread\_exit(NULL); // Exit the producer thread

}

// Consumer function: counts up from 1 to 10

void \*consumer(void \*param) {

int i;

for (i = 1; i <= 10; i++) {

sem\_wait(&s2); // Wait for the producer to finish counting down

printf("Consumer (Thread B) counts up: %d\n", i);

sleep(1); // Simulate work

sem\_post(&s1); // Signal the producer to continue counting down

}

pthread\_exit(NULL); // Exit the consumer thread

}

int main() {

pthread\_t thread1, thread2;

// Initialize semaphores

sem\_init(&s1, 0, 1); // Producer starts first, so s1 starts with 1

sem\_init(&s2, 0, 0); // Consumer waits for producer, so s2 starts with 0

// Create producer and consumer threads

pthread\_create(&thread1, NULL, producer, NULL);

pthread\_create(&thread2, NULL, consumer, NULL);

// Wait for threads to finish

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

// Destroy semaphores

sem\_destroy(&s1);

sem\_destroy(&s2);

return 0;

}

**Output**

srikant@SRIKANTs-PC:~/DOS\_2241016519/ass5$ ./q4

Thread A counts down: 10

Thread B counts up: 1

Thread A counts down: 9

Thread B counts up: 2

Thread A counts down: 8

Thread B counts up: 3

Thread A counts down: 7

Thread B counts up: 4

Thread A counts down: 6

Thread B counts up: 5

Thread A counts down: 5

Thread B counts up: 6

Thread A counts down: 4

Thread B counts up: 7

Thread A counts down: 3

Thread B counts up: 8

Thread A counts down: 2

Thread B counts up: 9

Thread A counts down: 1

Thread B counts up: 10

**Q5.**

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#define NUM\_ITERATIONS 7 // Each thread will print 7 numbers

sem\_t semA, semB, semC;

void \*A(void \*param) {

for (int i = 1; i <= NUM\_ITERATIONS; i++) {

sem\_wait(&semA); // Wait for signal to print

printf("A%d\n", (i - 1) \* 3 + 1); // Print A1, A4, A7, ...

sleep(1); // Simulate work

sem\_post(&semB); // Signal thread B to print

}

pthread\_exit(NULL);

}

void \*B(void \*param) {

for (int i = 1; i <= NUM\_ITERATIONS; i++) {

sem\_wait(&semB); // Wait for signal to print

// Special case for the last B (B20)

if (i == NUM\_ITERATIONS) {

printf("B20\n");

} else {

printf("B%d\n", (i - 1) \* 3 + 2); // Print B2, B5, B8, ...

}

sleep(1); // Simulate work

sem\_post(&semC); // Signal thread C to print

}

pthread\_exit(NULL);

}

void \*C(void \*param) {

for (int i = 1; i <= NUM\_ITERATIONS - 1; i++) { // Only print 6 values of C

sem\_wait(&semC); // Wait for signal to print

printf("C%d\n", (i - 1) \* 3 + 3); // Print C3, C6, C9, ...

sleep(1); // Simulate work

sem\_post(&semA); // Signal thread A to print

}

pthread\_exit(NULL);

}

int main() {

pthread\_t threadA, threadB, threadC;

// Initialize semaphores

sem\_init(&semA, 0, 1); // Thread A starts first

sem\_init(&semB, 0, 0); // Thread B waits

sem\_init(&semC, 0, 0); // Thread C waits

// Create threads

pthread\_create(&threadA, NULL, A, NULL);

pthread\_create(&threadB, NULL, B, NULL);

pthread\_create(&threadC, NULL, C, NULL);

// Wait for threads to finish

pthread\_join(threadA, NULL);

pthread\_join(threadB, NULL);

pthread\_join(threadC, NULL);

// Destroy semaphores

sem\_destroy(&semA);

sem\_destroy(&semB);

sem\_destroy(&semC);

return 0;

}

**Output:**

srikant@SRIKANTs-PC:~/DOS\_2241016519/ass5$ ./q5

A1

B2

C3

A4

B5

C6

A7

B8

C9

A10

B11

C12

A13

B14

C15

A16

B17

C18

A19

B20