1 /\*

1. Name : Darshan Samadhan Suryawanshi
2. Roll No : 50 (B)
3. Problem Statement : Write a program to solve Classical Problems of Synchronization using Mutex and Semaphore.
4. \*/

6

1. import java.util.LinkedList;
2. import java.util.concurrent.Semaphore;

9

1. class ProducerConsumerWithMutex {
2. LinkedList<Integer> buffer = new LinkedList<>();
3. int capacity = 5;
4. final Object mutex = new Object(); 14
5. public void produce() throws InterruptedException {
6. int value = 0;
7. while (true) {
8. synchronized (mutex) {
9. while (buffer.size() == capacity) {
10. mutex.wait(); // Wait if buffer is full

21 }

1. buffer.add(value);
2. System.out.println("Producer produced: " + value);
3. value++;
4. mutex.notify(); // Notify consumer

26 }

27 Thread.sleep(1000);

28 }

29 }

30

1. public void consume() throws InterruptedException {
2. while (true) {
3. synchronized (mutex) {
4. while (buffer.isEmpty()) {
5. mutex.wait(); // Wait if buffer is empty

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 36 |  |  |  | } |  |
| 37 |  |  |  | int val = buffer.removeFirst(); |
| 38 |  |  |  | System.out.println("Consumer consumed: " | + val); |
| 39 |  |  |  | mutex.notify(); // Notify producer |  |
| 40 |  |  |  | } |  |
| 41 |  |  |  | Thread.sleep(1000); |  |
| 42 |  |  | } |  |  |
| 43 |  | } |  | | |
| 44 | } |  |
| 45 |  |  |
| 46 |  |  |

1. class ProducerConsumerWithSemaphore {
2. LinkedList<Integer> buffer = new LinkedList<>();
3. int capacity = 5;
4. Semaphore empty = new Semaphore(5); // Semaphore for empty slots
5. Semaphore full = new Semaphore(0); // Semaphore for full slots
6. Semaphore mutex = new Semaphore(1); // Semaphore for mutual exclusion 53
7. public void produce() throws InterruptedException {
8. int value = 0;
9. while (true) {
10. empty.acquire(); // Wait for an empty slot
11. mutex.acquire(); // Lock the buffer
12. buffer.add(value);

60

61

62

63

64

65

66 }

67

System.out.println("Producer produced: " + value); value++;

mutex.release(); // Unlock the buffer

full.release(); // Signal that buffer is not empty Thread.sleep(1000);

}

68

69

70

71

72

73

74

75

76

77

78

79 }

80

public void consume() throws InterruptedException { while (true) {

full.acquire(); // Wait for a full slot mutex.acquire(); // Lock the buffer

int val = buffer.removeFirst();

System.out.println("Consumer consumed: " + val); mutex.release(); // Unlock the buffer

empty.release(); // Signal that buffer is not full

Thread.sleep(1000);

}

}

81 public class SynchronizationComparison { 82

1. public static void main(String[] args) throws InterruptedException {
2. // Test for Mutex
3. ProducerConsumerWithMutex pcMutex = new ProducerConsumerWithMutex();
4. Thread producerMutex = new Thread(() -> {
5. try {
6. pcMutex.produce();
7. } catch (InterruptedException e) {
8. Thread.currentThread().interrupt();

91 }

92 });

93

1. Thread consumerMutex = new Thread(() -> {
2. try {
3. pcMutex.consume();
4. } catch (InterruptedException e) {
5. Thread.currentThread().interrupt();

99 }

100 });

101

1. // Test for Semaphore
2. ProducerConsumerWithSemaphore pcSemaphore = new ProducerConsumerWithSemaphore();

|  |  |  |
| --- | --- | --- |
| 104 | Thread producerSemaphore = new Thread(() -> | { |
| 105 | try { |  |
| 106 | pcSemaphore.produce(); |  |
| 107 | } catch (InterruptedException e) { |  |
| 108 | Thread.currentThread().interrupt(); |  |
| 109 | } |  |
| 110 | }); |  |
| 111 |  |  |
| 112 | Thread consumerSemaphore = new Thread(() -> | { |
| 113 | try { |  |
| 114 | pcSemaphore.consume(); |  |
| 115 | } catch (InterruptedException e) { |  |
| 116 | Thread.currentThread().interrupt(); |  |
| 117 | } |  |
| 118 | }); |  |

119

120

121

122

123

124

125

126

127

128

129 }

130 }

131 /\*

System.out.println("Running Producer-Consumer with Mutex..."); producerMutex.start();

consumerMutex.start(); Thread.sleep(10000);

System.out.println("Running Producer-Consumer with Semaphore..."); producerSemaphore.start();

consumerSemaphore.start(); Thread.sleep(10000);

1. PS C:\Aditya\OneDrive\Desktop\SPOSPR> & 'C:\Program Files\Java\jdk- 20\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'

'C:\Aditya\AppData\Roaming\Code\User\workspaceStorage\b2aceacb3cce411dea48f538

f584f38a\redhat.java\jdt\_ws\SPOSPR\_3eb41cf5\bin' 'SynchronizationComparison'

1. Running Producer-Consumer with Mutex...
2. Producer produced: 0
3. Consumer consumed: 0
4. Producer produced: 1
5. Consumer consumed: 1
6. Producer produced: 2
7. Consumer consumed: 2
8. Producer produced: 3
9. Consumer consumed: 3
10. Producer produced: 4
11. Consumer consumed: 4
12. Producer produced: 5
13. Consumer consumed: 5
14. Producer produced: 6
15. Consumer consumed: 6
16. Producer produced: 7
17. Consumer consumed: 7
18. Producer produced: 8
19. Consumer consumed: 8
20. Producer produced: 9
21. Consumer consumed: 9
22. Running Producer-Consumer with Semaphore...
23. Producer produced: 0
24. Consumer consumed: 0
25. Producer produced: 10
26. Consumer consumed: 10
27. Producer produced: 1
28. Consumer consumed: 1
29. Producer produced: 11
30. Consumer consumed: 11
31. Producer produced: 2
32. Consumer consumed: 2
33. Producer produced: 12
34. Consumer consumed: 12
35. Producer produced: 3
36. Consumer consumed: 3
37. Producer produced: 13
38. Consumer consumed: 13
39. Producer produced: 4
40. Consumer consumed: 4
41. Producer produced: 14
42. Consumer consumed: 14
43. Producer produced: 5

|  |  |  |  |
| --- | --- | --- | --- |
| 176 | Consumer | consumed: | 5 |
| 177 | Producer | produced: | 15 |
| 178 | Consumer | consumed: | 15 |
| 179 | \*/ |  |  |