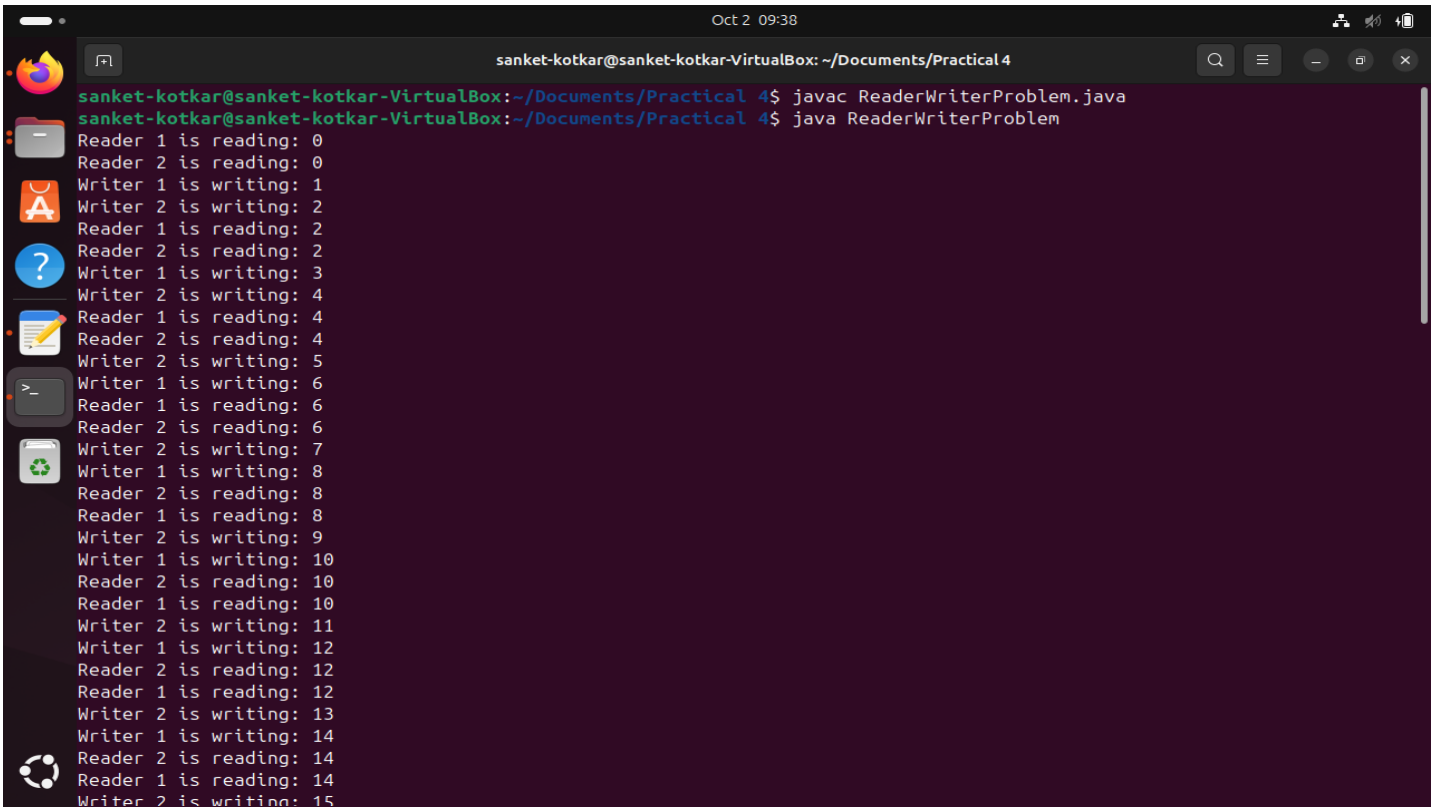
Program Code –

```
import java.util.concurrent.Semaphore;


class SharedData {
   int data = 0; // Shared resource
   int readerCount = 0; // Number of active readers


   // Semaphores
   Semaphore mutex = new Semaphore(1); // Protect readerCount
   Semaphore wrt = new Semaphore(1);   // Protect shared data (writers)
}


// Reader class
class Reader extends Thread {
   SharedData shared;
   int readerId;


   Reader(SharedData shared, int id) {
      this.shared = shared;
      this.readerId = id;
   }


   public void run() {
      try {
         while (true) {
            // Entry section
            shared.mutex.acquire();
            shared.readerCount++;
            if (shared.readerCount == 1)
               shared.wrt.acquire(); // First reader locks writer
            shared.mutex.release();


            // Critical section
            System.out.println("Reader " + readerId + " is reading: " + shared.data);
            Thread.sleep(500); // Simulate reading time


            // Exit section
            shared.mutex.acquire();
            shared.readerCount--;
            if (shared.readerCount == 0)
               shared.wrt.release(); // Last reader releases writer
            shared.mutex.release();


            Thread.sleep(500); // Simulate other work
         }
      } catch (InterruptedException e) {
         System.out.println("Reader interrupted");
      }
   }
}


// Writer class
class Writer extends Thread {
   SharedData shared;
   int writerId;


   Writer(SharedData shared, int id) {
      this.shared = shared;
      this.writerId = id;
   }


   public void run() {
```

```
        try {
            while (true) {
                shared.wrt.acquire(); // Only writer
access
                shared.data += 1;     // Modify shared
data
                System.out.println("Writer " + writerId
+ " is writing: " + shared.data);
                Thread.sleep(1000);   // Simulate writing
time
                shared.wrt.release();

                Thread.sleep(500); // Simulate other
work
            }
        } catch (InterruptedException e) {
            System.out.println("Writer interrupted");
        }
    }
}
```

```
// Main class
public class ReaderWriterProblem {
    public static void main(String[] args) {
        SharedData shared = new SharedData();

        // Create readers and writers
        Reader r1 = new Reader(shared, 1);
        Reader r2 = new Reader(shared, 2);
        Writer w1 = new Writer(shared, 1);
        Writer w2 = new Writer(shared, 2);

        // Start threads
        r1.start();
        r2.start();
        w1.start();
        w2.start();
    }
}
```

Output –

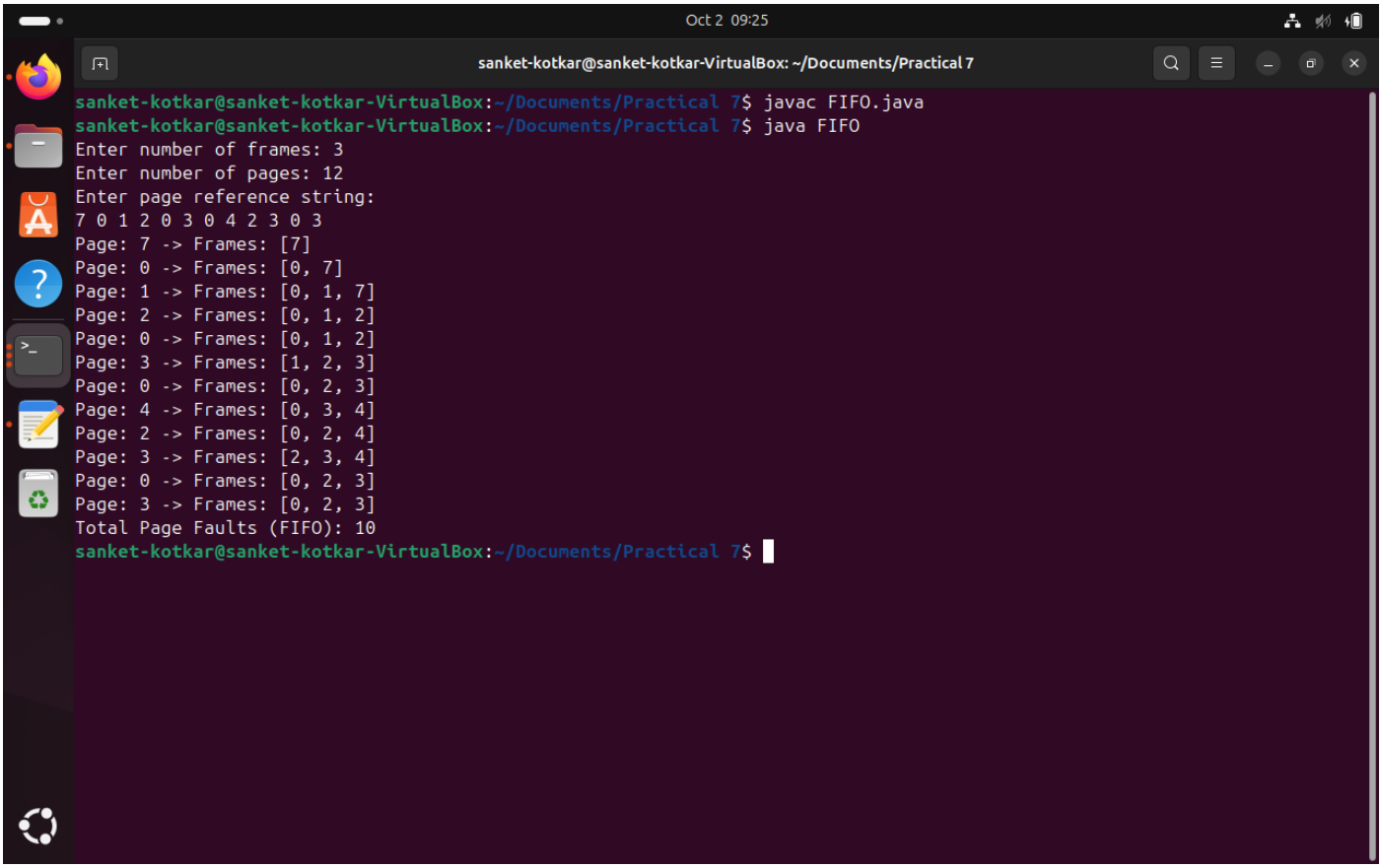


1. **FIFO Page Replacement -**

Program Code –

```
import java.util.*;

public class FIFO {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter number of frames: ");
      int frames = sc.nextInt();

      System.out.print("Enter number of pages: ");
      int n = sc.nextInt();
      int[] pages = new int[n];

      System.out.println("Enter page reference string:");
      for (int i = 0; i < n; i++) {
         pages[i] = sc.nextInt();
      }

      Queue<Integer> queue = new LinkedList<>();
      Set<Integer> set = new HashSet<>();
      int faults = 0;

      for (int p : pages) {
         if (!set.contains(p)) {
            if (set.size() < frames) {
               set.add(p);
               queue.add(p);
            } else {
               int removed = queue.poll();
               set.remove(removed);
               set.add(p);
               queue.add(p);
            }
            faults++;
         }
         System.out.println("Page: " + p + " -> Frames: " + set);
      }

      System.out.println("Total Page Faults (FIFO): " + faults);
      sc.close();
   }
}
```

Output –



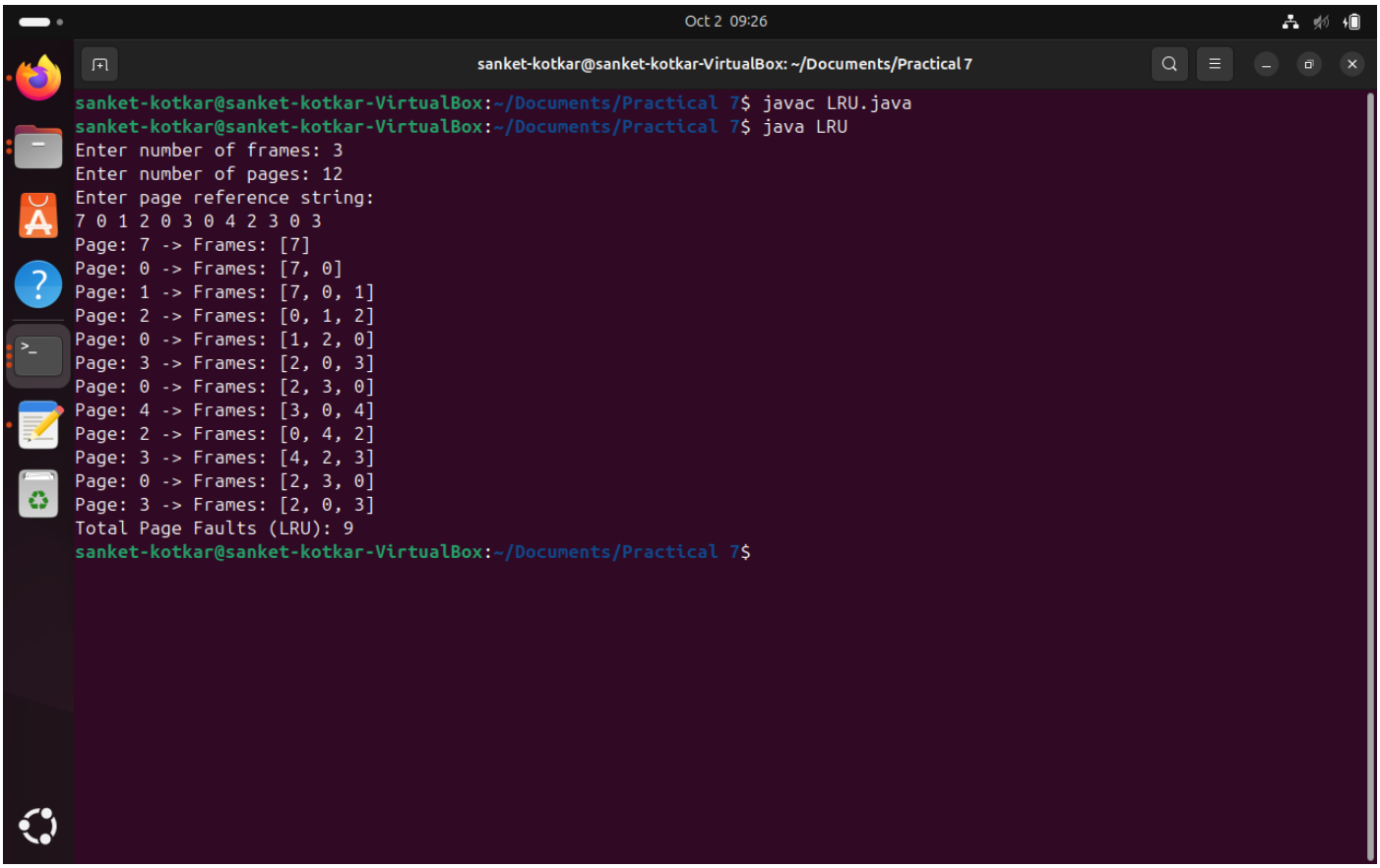
2. LRU Page Replacement –

Program Code –

```
import java.util.*;

public class LRU {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of frames: ");
        int frames = sc.nextInt();

        System.out.print("Enter number of pages: ");
        int n = sc.nextInt();
        int[] pages = new int[n];

        System.out.println("Enter page reference string:");
        for (int i = 0; i < n; i++) {
            pages[i] = sc.nextInt();
        }

        List<Integer> list = new ArrayList<>();
        Set<Integer> set = new HashSet<>();
        int faults = 0;

        for (int p : pages) {
            if (!set.contains(p)) {
                if (set.size() < frames) {
                    set.add(p);
                    list.add(p);
                } else {
                    int lru = list.remove(0);
                    set.remove(lru);
                    set.add(p);
```

```
            list.add(p);
        }
        faults++;
    } else {
        list.remove((Integer) p);
        list.add(p);
    }
    System.out.println("Page: " + p + " -> Frames: " + list);
}
System.out.println("Total Page Faults (LRU): " + faults);
sc.close();
}
}
```

Output –



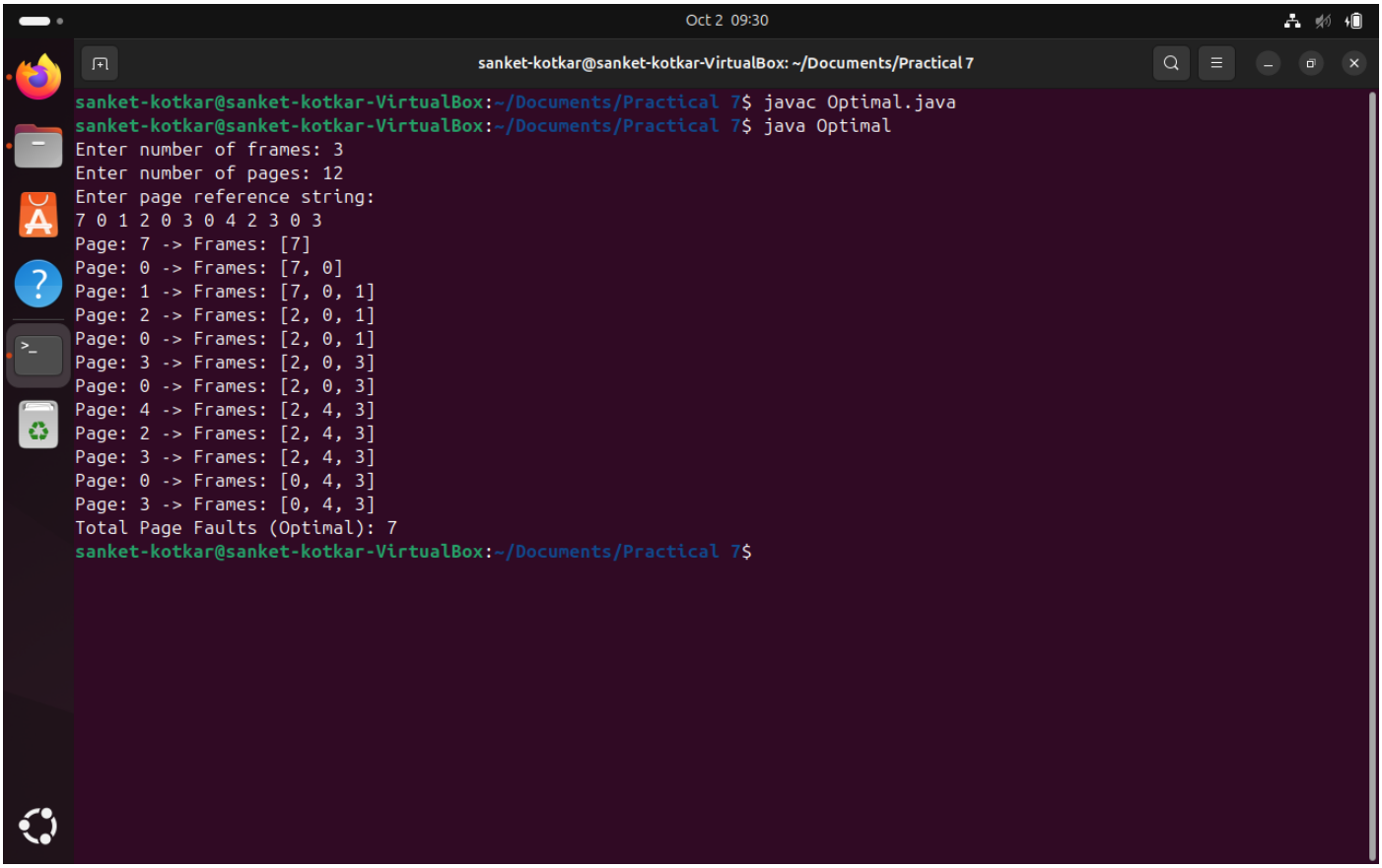
3. Optimal Page Replacement –

Program Code –

```
import java.util.*;

public class Optimal {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of frames: ");
        int frames = sc.nextInt();
        System.out.print("Enter number of pages: ");
```

```
        int n = sc.nextInt();
        int[] pages = new int[n];

        System.out.println("Enter page reference string:");
        for (int i = 0; i < n; i++) {
            pages[i] = sc.nextInt();
        }

        List<Integer> memory = new ArrayList<>();
        int faults = 0;

        for (int i = 0; i < pages.length; i++) {
            int p = pages[i];

            if (!memory.contains(p)) {
                if (memory.size() < frames) {
                    memory.add(p);
                } else {
                    int farthest = -1, indexToReplace = -1;
                    for (int j = 0; j < memory.size(); j++) {
                        int page = memory.get(j);
                        int nextUse = Integer.MAX_VALUE;
                        for (int k = i + 1; k < pages.length; k++) {
                            if (pages[k] == page) {
                                nextUse = k;
                                break;
                            }
                        }
                        if (nextUse > farthest) {
                            farthest = nextUse;
                            indexToReplace = j;
                        }
                    }
                    memory.set(indexToReplace, p);
                }
                faults++;
            }
            System.out.println("Page: " + p + " -> Frames: " + memory);
        }

        System.out.println("Total Page Faults (Optimal): " + faults);
        sc.close();
    }
}
```

Output -


Oct 2 09:30
sanket-kotkar@sanket-kotkar-VirtualBox: ~/Documents/Practical 7
sanket-kotkar@sanket-kotkar-VirtualBox:~/Documents/Practical 7$ javac Optimal.java
sanket-kotkar@sanket-kotkar-VirtualBox:~/Documents/Practical 7$ java Optimal
Enter number of frames: 3
Enter number of pages: 12
Enter page reference string:
7 0 1 2 0 3 0 4 2 3 0 3
Page: 7 -> Frames: [7]
Page: 0 -> Frames: [7, 0]
Page: 1 -> Frames: [7, 0, 1]
Page: 2 -> Frames: [2, 0, 1]
Page: 0 -> Frames: [2, 0, 1]
Page: 3 -> Frames: [2, 0, 3]
Page: 0 -> Frames: [2, 0, 3]
Page: 4 -> Frames: [2, 4, 3]
Page: 2 -> Frames: [2, 4, 3]
Page: 3 -> Frames: [2, 4, 3]
Page: 0 -> Frames: [0, 4, 3]
Page: 3 -> Frames: [0, 4, 3]
Total Page Faults (Optimal): 7
sanket-kotkar@sanket-kotkar-VirtualBox:~/Documents/Practical 7$