```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <thread>
#include <chrono>

using namespace std;

// Structure to represent an item with value, weight, and index
struct Item {
    int value;
    int weight;
    int index;
    double ratio; // Value to weight ratio
};

// Comparator function to sort items based on their value-to-weight ratio in descending order
bool compare(Item a, Item b) {
    return a.ratio > b.ratio;
}

// Function to clear the screen for visualizing steps
void clearScreen() {
    #ifdef _WIN32
        system("cls");
    #else
        system("clear");
    #endif
}

// Function to display items and their status at each step
void displayItems(const vector<Item>& items, int knapsackWeight, double totalValue) {
    cout << "Knapsack Weight: " << knapsackWeight << endl;
    cout << "Total Value: " << totalValue << endl << endl;

    cout << "Items in the knapsack:" << endl;
    for (const auto& item : items) {
        cout << "Item " << item.index + 1 << " | Value: " << item.value << " | Weight: " << item.weight
             << " | Ratio: " << item.ratio << endl;
    }
    cout << endl;
}

// Function to solve the Fractional Knapsack problem and visualize the process
void fractionalKnapsackVisualization(vector<Item>& items, int capacity) {
    // Sort items based on their value/weight ratio in descending order
    for (auto& item : items) {
        item.ratio = (double)item.value / item.weight;
    }
    sort(items.begin(), items.end(), compare);

    int knapsackWeight = 0;
    double totalValue = 0.0;
    clearScreen();

    for (const auto& item : items) {
        if (knapsackWeight + item.weight <= capacity) {
            // If the full item fits, take it
            knapsackWeight += item.weight;
            totalValue += item.value;
            clearScreen();
            displayItems(items, knapsackWeight, totalValue);
            cout << "Full item " << item.index + 1 << " taken!" << endl;
            std::this_thread::sleep_for(std::chrono::seconds(1));
        } else {
            // Take the fraction of the item that fits
            int remainingWeight = capacity - knapsackWeight;
            double fraction = (double)remainingWeight / item.weight;
            knapsackWeight += remainingWeight;
            totalValue += item.value * fraction;
            clearScreen();
            displayItems(items, knapsackWeight, totalValue);
            cout << "Fraction of item " << item.index + 1 << " taken: "
                 << fraction * 100 << "%" << endl;
            break; // After fractional inclusion, the knapsack is full
        }
    }
}

// Main function
int main() {
    int n, capacity;

    cout << "Enter number of items: ";
    cin >> n;

    vector<Item> items(n);

    cout << "Enter capacity of the knapsack: ";
    cin >> capacity;

    for (int i = 0; i < n; ++i) {
        cout << "Enter value and weight of item " << i + 1 << ": ";
        cin >> items[i].value >> items[i].weight;
        items[i].index = i;
    }

    fractionalKnapsackVisualization(items, capacity);

    return 0;
}

```