BÀI THỰC HÀNH SỐ 3 THAO TÁC NGUYÊN TỐ VÀ GIAO DỊCH

Như đã biết, với thẻ thông minh luôn có nguy cơ xảy ra lỗi bất cứ lúc nào trong quá trình thực thi Applet. Lỗi có thể là lỗi tính toán, hoặc thường xuyên hơn là lỗi người sử dụng vô tình gỡ thẻ ra khỏi CAD, cắt nguồn cung cấp cho CPU thẻ và thực thi bất kì một Applet nào. Việc rút thẻ khỏi CAD được gọi là xé hoặc xé thẻ. Việc thực thi không đầy đủ đưa ra một thách thức cho việc duy trì tính toàn vẹn của các hoạt động dữ liệu trên dữ liệu nhạy cảm trong thẻ thông minh.

JCRE cung cấp một cơ chế mạnh mẽ để đảm bảo hoạt động nguyên tử. Cơ chế này hoạt động ở hai cấp độ.

- Thứ nhất, nền tảng Java Card bảo đảm rằng bất kì mọi cập nhật đến một trường đơn trong đối tượng tồn tại liên tục hoặc là lớp đơn (class) của các trường là nguyên tử.
- Thứ hai, nền tảng Java Card hỗ trợ mô hình giao dịch, trong đó một Applet có thể nhóm một nhóm các bản cập nhật thành một giao dịch. Trong mô hình này, tính nguyên tử của tất cả các bản cập nhật được đảm bảo.

3.1 Thao tác nguyên tố

Trên nền tảng Java Card, tính nguyên tử có nghĩa là mọi cập nhật cho một trường của đối tượng liên tục đơn lẻ (bao gồm cả các phần tử mảng) hoặc cho lớp của trường được đảm bảo thành công hoàn toàn hoặc nếu không thành công (trường hợp xảy ra lỗi trong quá trình cập nhật) thì sẽ khôi phục về giá trị ban đầu. Ví dụ, trường của đối tượng hiện tại đang chứa giá trị là 1 và nó đang bắt đầu cập nhật với giá trị là 2. Vô tình thẻ bị giật ra khỏi CAD vào thời điểm quan trọng khi thẻ đang ghi đè lên trường. Khi có điện trở lại, trường sẽ không nhận giá trị ngẫu nhiên mà giá trị sẽ được khôi phục về giá trị trước đó, có nghĩa là bằng 1.

Khái niệm nguyên tử chỉ áp dụng cho các nội dung của bộ nhớ liên tục. Tính năng nguyên tử của JCRE không áp dụng cho các mảng tạm thời. Việc cập nhật một phần tử của mảng tạm thời không bảo toàn giá trị trước đó trong trường hợp mất điện. Trong lần tiếp theo thẻ được đưa vào CAD, các phần tử của mảng tạm thời được đặt thành giá trị mặc định của chúng (`zero`, `false` hoặc là `null`).

3.2 Giao dịch

Tính nguyên tử bảo đảm việc sửa đổi nguyên tử của một đối tượng dữ liệu đơn lẻ. Tuy nhiên, một Applet có thể cần cập nhật nguyên tử một số trường khác nhau trên các đối tượng khác nhau. Ví dụ giao dịch tín dụng hoặc giao dịch ghi nợ có thể yêu cầu ví Applet làm tăng số giao dịch, cập nhật số dư tài khoản và ghi vào nhật ký giao dịch.

Người đọc có thể quen với việc sử dụng các khái niệm giao dịch cơ sở dữ liệu như bắt đầu, commit và khôi phục để bảo đảm rằng việc cập nhật cho nhiều giá trị hoặc là hoàn thành hoặc là không gì cả. Công nghệ Java Card hỗ trợ một mô hình giao dịch tương tự, với khả năng commit và khôi phục để đảm bảo rằng các hoạt động phức tạp có thể được thực hiện nguyên tử, hoặc là chúng thành công, hoặc là một phần kết quả không có ý nghĩa. Cơ chế giao dịch bảo vệ khỏi các sự kiện như mất điện ở giữa giao dịch hay các lỗi chương trình có thể gây ra sự thay đổi dữ liệu nếu tất cả các bước của giao dịch không kết thúc theo cách thông thường.

Ở đây, chúng ta có thể giải thích khái niệm giao dịch trong công nghệ Java Card như sau. Như chúng ta đã biết, thẻ thông minh có hai loại bộ nhớ là: bộ nhớ liên tục (EEPROM hoặc Flash) và bộ nhớ tạm thời (RAM). Khi bị ngắt nguồn cung cấp, tất cả dữ liệu lưu trữ trong bộ nhớ tạm thời (RAM) sẽ bị mất và chỉ có dữ liệu lưu trữ trong bộ nhớ liên tục được bảo tồn. Theo mặc định, tất cả các đối tượng và trường của thẻ thông minh thường được lưu trong bộ nhớ liên tục. Vì vậy, nếu thẻ thông minh bị ngắt nguồn đột ngột trong quá trình cập nhập dữ liệu trong bộ nhớ liên tục, sẽ làm cho thẻ thông minh rơi vào trạng thái không nhất quán (nghĩa là dữ liệu có thể được cập nhật không theo ý muốn). Để đối phó với hiện tượng

này, nền tảng Java Card cung cấp một cơ chế được gọi là giao dịch. Nó đảm bảo rằng một số cập nhật trong bộ nhớ liên tục được thực thi dưới dạng một hoạt động nguyên tử (tức là tất cả các cập nhật đều được thực hiện hoặc không gì cả).

Nền tảng Java Card cung cấp 3 phương thức giao dịch là:

- `JCSystem.beginTransaction`
- `JCSystem.commitTransaction`
- `JCSystem.abortTransaction`.

Ví dụ 1: Cho `u` và `t` là hai mảng có cùng độ dài, được lưu trong bộ nhớ EEPROM, giả sử cần sao chép các phần tử của `u` sang `t`.

```
JCSystem.beginTransaction();
for(int i=0; i++; i<t.length)
   t[i] = u[i];
JCSystem.commitTransaction();
```

Việc sao chép này được thực hiện dưới dạng một hoạt động nguyên tử. Nghĩa là, nếu thẻ thông minh bị ngắt nguồn đột ngột khi thực thi đoạn code trên, thì hoặc là tất cả các phần tử của `u` được sao chép sang `t`, hoặc không phần tử nào được sao chép.

Ví dụ 2: Cho có 2 đối tượng tồn tại liên tục là mảng `buffer1` và `buffer2` có cùng độ dài là 32, giả sử cần sao chép các phần tử của `buffer1` sang `buffer2`

```
JCSystem.beginTransaction();
for (short i = 0; i < 32; i++)
{
     buffer2[i] = buffer1[i];
     if (i ==(short)5)
     {
          JCSystem.abortTransaction();
```

```
                JCSystem.beginTransaction(); // bat dau mot giao
dich moi voi i > 5
           }
      }
      JCSystem.commitTransaction();
```

Trong ví dụ này, khi i = 5 giao dịch sẽ bị hủy bỏ, nghĩa là việc giao dịch sao chép các phần tử của mảng `buffer1` sang `buffer2` với $i \leq 5$ sẽ bị hủy bỏ, và giá trị các phần tử của `buffer2` với $i \leq 5$ sẽ được cập nhật về giá trị trước đó. Các giao dịch sao chép các phần tử còn lại của `buffer1` sang `buffer2` sẽ được thực hiện dưới dạng một hoạt động nguyên tử.

Để xem xét cách thức hoạt động của giao dịch trong nền tảng Java Card, chúng ta xây dựng một Applet với 2 lệnh chính: thực hiện giao dịch và gửi giá trị của `buffer2` lên máy chủ.

```
package bai3_Transaction;
import javacard.framework.*;
public class Applet1 extends Applet
{
      // khai bao ma INS trong tieu de cua apdu
      final static byte INS_TRANSACTION  = (byte)0x00;
      final static byte INS_SEND              = (byte)0x01;

      //khai bao cac doi tuong va bien
      private static byte[] buffer1, buffer2;

      public static void install(byte[] bArray, short bOffset,
byte bLength)
      {
           new Applet1().register(bArray, (short) (bOffset + 1),
bArray[bOffset]);
```

```
        buffer1 = new byte[32];
        buffer2 = new byte[32];
        for (short i = 0; i < 32; i++ )
            buffer1[i] = (byte)i;
    }

    public void process(APDU apdu)
    {
        if (selectingApplet())
        {
            return;
        }

        byte[] buf = apdu.getBuffer();
        apdu.setIncomingAndReceive();

        switch (buf[ISO7816.OFFSET_INS])
        {
        case INS_TRANSACTION:
            JCSystem.beginTransaction();
            for (short i = 0; i < 32; i++)
            {
                buffer2[i] = buffer1[i];
                if (i ==(short)5)
                {
                    JCSystem.abortTransaction();
                    JCSystem.beginTransaction();// bat dau
mot giao dich moi voi i > 5
                }
            }
            JCSystem.commitTransaction();
```

```
                break;
            case INS_SEND:
                apdu.setOutgoing();
                apdu.setOutgoingLength((short)32);
                apdu.sendBytesLong(buffer2, (short)0, (short)32);
                break;
            default:

        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
            }
        }
}
```

Kết quả thực thi Applet trên như sau:

```
>> /send 00010102
>> 00 01 01 02
<< 00 00 00 00 00 00 00 00 00 00 90 00

>> /send 00000102
>> 00 00 01 02
<< 90 00

>> /send 00010102
>> 00 01 01 02
<< 00 00 00 00 00 00 06 07 08 09 90 00
```

Kết quả trên cho thấy các phần tử mảng buffer2 với $i \leq 5$ được cập nhật về giá trị ban đầu, chỉ có các phần tử còn lại được cập nhật sang giá trị mới.

3.3 Biến cục bộ và đối tượng tạm thời trong giao dịch

Chú ý rằng chỉ những cập nhật cho các đối tượng liên tục mới được tham gia vào giao dịch. Việc cập nhật các đối tượng tạm thời và các biến cục bộ (bao gồm tham số của phương thức) không bao giờ được hoàn tác bất kể chúng có ở "bên

trong" một giao dịch hay không. Các biến cục bộ được tạo ra trong ngăn xếp Java Card nằm trong RAM.

Ví dụ: Cho đối tượng liên tục buffer1, đối tượng tạm thời buffer2 và biến cục bộ a_local, giả sử việc cập nhật giá trị của các đối tượng và biến đều được đặt trong một giao dịch, giá trị của các đối tượng và biến sau đó được gửi lên máy chủ.

```
public void process(APDU apdu)
     {
          if (selectingApplet())
          {
               return;
          }

          byte[] buf = apdu.getBuffer();
          apdu.setIncomingAndReceive();

          switch (buf[ISO7816.OFFSET_INS])
          {
          case (byte)0x00:
               byte[] buffer1 =  new byte[3];
               byte[] buffer2 =
JCSystem.makeTransientByteArray((short)3,
JCSystem.CLEAR_ON_RESET);

               JCSystem.beginTransaction();
               for (byte i = 0; i < 3; i++) {
                 buffer1[i] = (byte)(i+1);
                 buffer2[i] = (byte)(2*(byte)(i+1));
               }
               byte a_local = 1;
               JCSystem.abortTransaction();
```

```
                apdu.setOutgoing();
                apdu.setOutgoingLength((short)7);
                apdu.sendBytesLong(buffer1, (short)0, (short)3);
                apdu.sendBytesLong(buffer2, (short)0, (short)3);
                Util.arrayFillNonAtomic(buf, (short)0, (short)1,
a_local);
                apdu.sendBytes((short)0, (short)1);
                break;
           default:

     ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
           }
     }
```

Kết quả thực thi Applet trên như sau:

```
>> /send 00000102
>> 00 00 01 02
<< 00 00 00 02 04 06 01 90 00
```

Kết quả này cho ta thấy, khi giao dịch bị hủy bỏ, chỉ có đối tượng liên tục được cập nhật về giá trị ban đầu, còn đối tượng tạm thời và biến cục bộ vẫn giữ nguyên giá trị mới.

Như vậy có thể kết luận, chỉ có đối tượng liên tục mới tham gia vào giao dịch, còn đối tượng tạm thời và biến cục bộ không tham gia vào giao dịch.